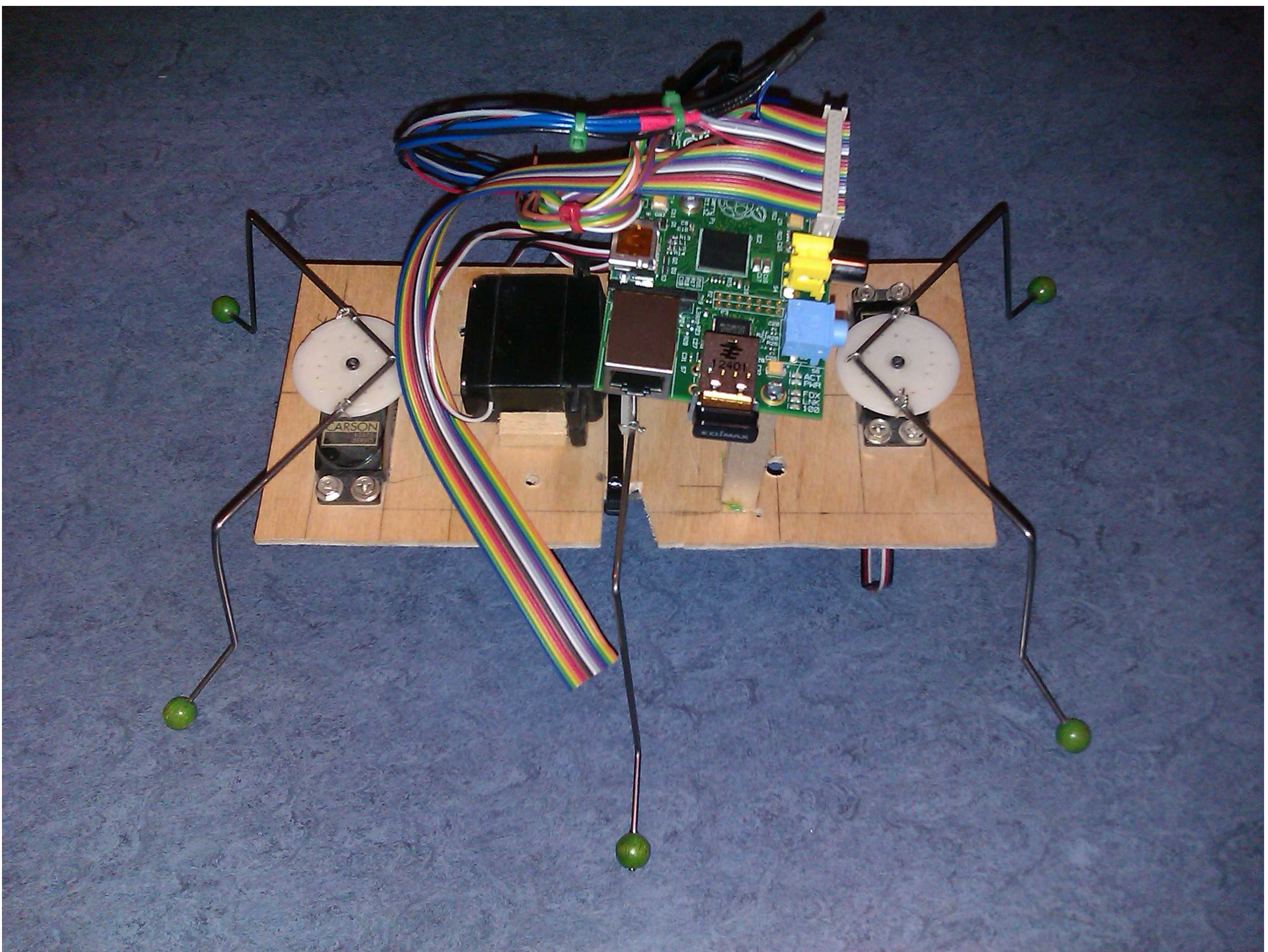
Bau und Programmierung einer Raspberry Pi Roboter-Ameise
Mit einer Einführung in LINUX und c/c++



1 Einleitung

In diesem Buch wird beschrieben, wie wir mit dem kleinen LINUX-Rechner und ein paar Servos[1] aus dem Modellbau eine Roboter-Ameise zum Laufen bringen. Diese Roboter-Ameise können wir dann autonom[2] laufen lassen oder mit einem alten Laptop fernsteuern. Als kleinen LINUX-Rechner verwenden wir hier den Raspberry Pi[3], mit dem sich ohne weitere Elektronik die Servos direkt ansteuern lassen.
Es reichen 3 Servos, die jeweils zwei Beine antreiben. Servos bekommt man schon für ca. 3,- € und der Raspberry Pi ist für ca. 30,-€ zu haben. Die Festplatte unseres LINUX-Rechners ist eine SDHC[4]-Karte mit mindestens 4 GB, die etwa 7,- € kostet. Wir brauchen auch noch einen Monitor mit DVI-[5] oder HDMI-Anschluss, sowie eine Tastatur und Maus jeweils mit USB-Anschluss. Ein 4-Port USB Hub mit Netzteil ist auch hilfreich. Der übernimmt dann gleich die Spannungsversorgung.
Als Gestell reicht ein kleines Sperrholzbrett, das wir uns zurecht sägen, oder wir stellen uns eine Platine her. Für die Funkverbindung brauchen wir noch einen WLAN-USB-Adapter, der für etwa 12,-€ zu haben ist. Kleinteile, wie Stecker, Kabel, Schrumpfschläuche[6], Lötzinn und der gleichen besorgen wir nach und nach. Es ist gut, wenn wir einen Elektronikhändler in der Nähe haben, sonst bestellen wir bei einem der Online-Händer, die im Anhang aufgelistet sind. Einige Kleinteile und vor allem Schrauben habe ich vom Ausschlachten alter Elektronik und manchem Kabel konnte ich so zu einem zweiten Leben verhelfen. Damit sollten unsere Kosten überschaubar bleiben. Für meine Roboter-Ameise rechnete ich nicht mal 60, € zusammen.
Etwas Werkzeug zu haben, ist auch kein Fehler. Ein Lötkolben ist nützlich und ein paar Schraubenzieher. Wenn ihr ein Messgerät habt, erleichtert das die Fehlersuche. Eine Laubsäge, um das Sperrholzbrettchen zu bearbeiten, ist auf jeden Fall von Vorteil.
In diesem Buch werden wir sämtliche Grundlagen besprechen, die notwendig sind um das Projekt Roboter-Ameise zu meistern, so dass wir am Ende nicht nur einen tollen Roboter haben, sondern auch etwas Programmieren und LINUX kennengelernt haben. Für die Fernsteuerung werden wir uns noch mit der Netzwerktechnik beschäftigen und dabei lernen, wie man aus einem alten Laptop einen prima WLAN-Server macht.
Was braucht man noch, um erfolgreich eine Roboter-Ameise zu bauen? Vor allem Geduld und Zähigkeit und den Willen niemals aufzugeben. Am meisten Spaß hatte ich immer, wenn ich unüberwindbar scheinende Probleme letztendlich gelöst habe. Mir hat der Bau meiner Roboter-Ameise viel Spaß gemacht und den möchte ich mit euch teilen und wünsche euch viel Erfolg beim Nachbau. Schickt mir ein Bild von eurem Ungetüm, wenn ihr es geschafft habt.

---

1 Ein Servo besteht aus einem elektrischen Motor mit Getriebe und einer elektronischen Ansteuerung und trotz seines geringen Preises ein mechatronisches Wunderwerk.

2 Autonom bedeutet, das der Roboter von allein laufen kann und z.B. Hindernissen selbst ausweicht.

3 Zu deutsch „Himbeerkuchen". Entwickelt wurde dieser Einplatinen-Computer von einer Wohltätigkeitsorganisation in England, die sich zum Ziel gesetzt hat, in Schulen die Informatik zu fördern. (siehe www.raspberrypi.org)

4 SDHC-Speicher-Karten sind der Standard vieler Kameras, Tablets, Netbooks und manchmal auch von Handys. SD steht für „secure digital", zu deutsch „sicher digital". HC steht für „high capacity", also hohe Speicherkapazität. Die Karten sind 2,4 x 3,2 cm² groß und 2,1 mm dick und haben 9 elektrische Kontakte. Die SDHC-Karten sind nach der Schreibgeschwindigkeit klassifiziert (Class 6 bedeutet 6 MB/s). SDHC-Karten sind nicht flüchtige Speicher, das heißt, sie behalten ihre Information, auch wenn keine Versorgungsspannung anliegt.

5 Haben wir am Monitor nur einen DVI-Eingang, so benötigen wir ein HDMI zu DVI Kabel. VGA unterstützt der Raspberry Pi leider nicht, dafür kann man aber versuchen den analogen Video-Ausgang mit einem Fernseher zu verbinden.

6 Schrumpfschläuche sind kleine Plastikröhrchen, die sich zusammenziehen, wenn es ihnen zu heiß wird.

Inhaltsverzeichnis

2 Die Roboter-Ameise

Eine mit einfachen Mitteln zusammengebaute Roboter-Ameise seht ihr in Bild 2. Sie hat sechs Beine, wie jede Ameise, und ein kleines Gehirn, das ist der Raspberry Pi LINUX-Computer. In diesem Kapitel wird die Montage beschrieben.

Der Raspberry Pi ist ein kleiner billiger LINUX-Rechner, mit dem wir uns später noch genauer befassen werden. Er läßt sich bei den Firmen RS oder Farnell übers Internet besorgen[7], ist aber häufiger ausverkauft. Es gibt zwei Modelle A und B. Das A Modell tut es für unsere Zwecke auch und kostet ca. 20,-€ plus Versandkosten. Das B Modell kostet ca. 10,- € mehr, hat aber dafür mehr Stecker-Möglichkeiten und einen größeren dynamischen[8] Speicher. Wir benötigen noch einen WLAN-USB-Adapter, einen USB-Hub[9], eine Tastatur und eine Maus[10]. Wenn wir das nicht noch irgendwo anders her besorgen, bekommen wir es auch bei RS oder Farnell und sparen Versandkosten.

Die sechs Beinchen werden von drei Modellbau-Servos bewegt, also immer zwei Beinchen werden von einem Servo bewegt. Wie Servos funktionieren und wie man sie anschließt, erkläre ich euch in Kapitel 10. Für ein Servo muss man etwa 6,-€ ausgeben[11] und bekoomt die zum Beispiel bei Conrad oder HobbyKing. Wir brauchen Servos mit den Abmessungen 40 x 20 mm².

2.1 Die Grundplatte mit den Servos

Als Grundplatte tut es ein Holzbrettchen, das etwa 100 x 200 mm² groß und ungefähr 3 bis 4 mm dick ist. So etwas kann man im Baumarkt kaufen. Ich habe versucht, die Kosten so gering wie möglich zu halten und bin dann bei einer alten Schublade vom Sperrmüll fündig geworden. In diese Grundplatte sägen wir uns mit der Laubsäge vorne und hinten zwei rechteckige Aussparungen von ca. 40 x 20 mm² für unsere Servos[12] und zwar so, dass die Drehachsen des vorderen und hinteren Servos auf der Mittellinie der Grundplatte sind.

Ich habe euch eine CAD-Zeichnung mit dem Programm FreeCAD[13] gezeichnet, wie ihr in Bild 1 erkennt. Die Datei der Zeichnung ist auch auf der beiliegenden CD zu finden. Falls die Servos nicht gut in die Aussparung passen sollten, kann man mit einer Nagelfeile etwas nachbessern. Die Position der Servos ist auch auf der Grundplatte in Bild 2 zu sehen. Die Servos werden mit vier Schrauben auf der Grundplatte befestigt.

7 http://raspberrypi.rsdelivers.com/default.aspx
http://piregistration.element14.com/raspberryPi1.html?none#raspberrypi

8 Ein dynamischer Speicher verliert seine Information, wenn die Spannungsversorgung fehlt. Dafür kann man aber sehr viel schneller Daten abspeichern und auslesen, als zum Beispiel mit der SDHC-Karte.

9 Ein USB-Hub ist eine Erweiterung eines USB-Steckers um typischer Weise 4 weitere USB-Steckplätze. Es gibt welche mit externem Netzteil. Solche können wir dann direkt zur Stromversorgung unsres Raspberry Pi benutzen.

10 Bei Maus und Tastatur tun es die billigsten. Wichtig ist nur, dass sie USB-Stecker haben.

11 http://www.conrad.de/ce/de/product/233751/Modelcraft-Standard-Servo-RS-2-Gleitlager-Getriebe-Kunststoff-JR
http://www.hobbyking.com/hobbyking/store/__3743__HXT_6_9kg_39_2g_16sec_Twin_bearing_servo.html

12 Die exakten Maße für die Aussparung entnehmen wir dem Datenblatt der Servos.

13 FreeCAD ist ein kostenloses 3d CAD Programm, dass auf den Betriebssystemen LINUX, Apple und Windows funktioniert. Es kann bei Sourceforeg unter http://sourceforge.net/projects/free-cad herunter geladen werden. Es wird von Jürgen Riegel und Werner Mayer betreut und spricht auch deutsch, wenn's denn sein muss. Es versteht die gängigen Datei-Formate für CAD-Zeichnungen wie DXF, SVG, STEP und IGES.

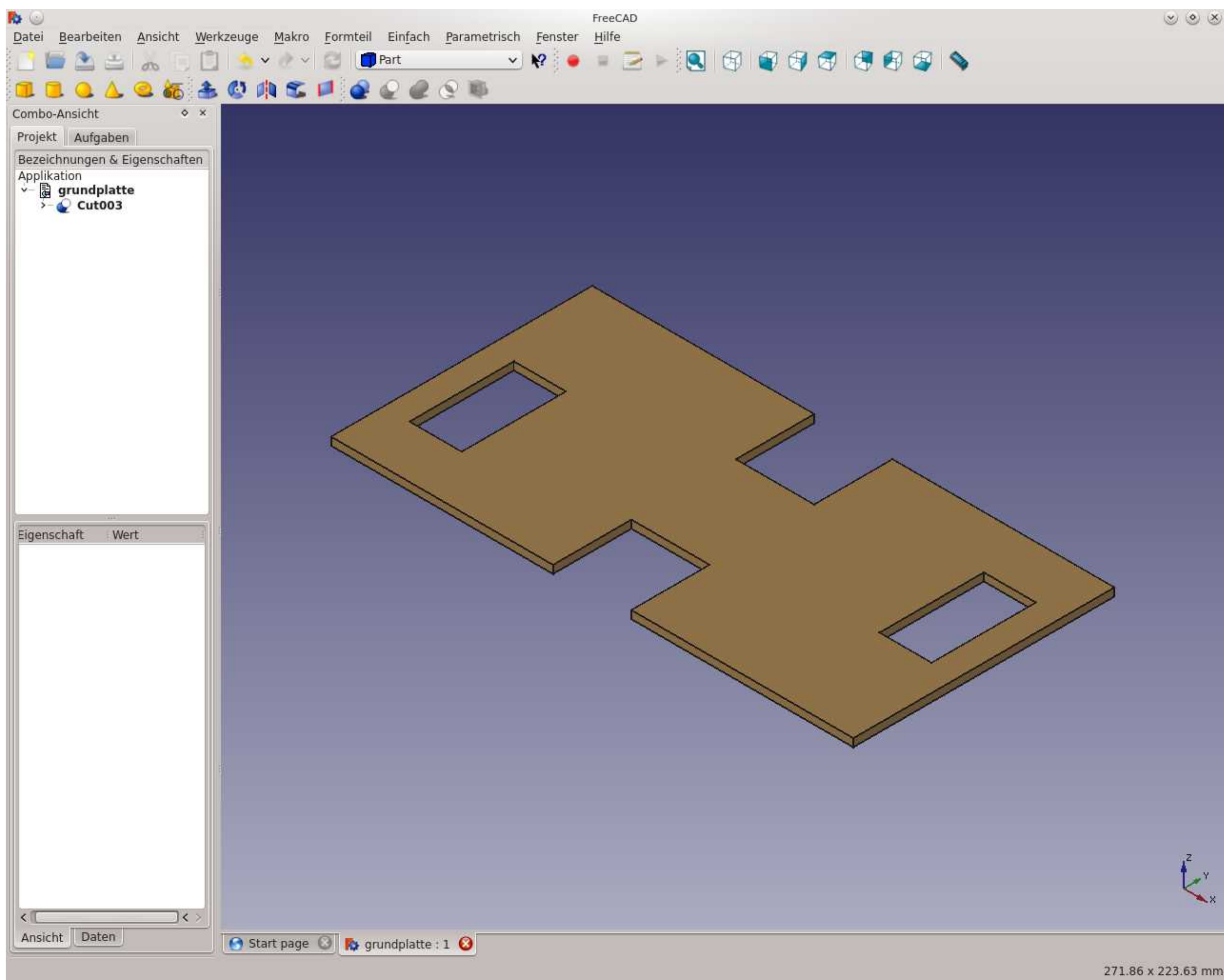


Abbildung 1: Die Sperrholz Grundplatte mit den Aussparungen für die Servos, gezeichnet mit dem Pogramm FreeCAD

Ich habe Schrauben mit einem großen Kopf genommen, die ich beim Ausschlachten eines alten Computers erbeutet habe. Nachdem ich die Servos probehalber in die Grundplatte gesetzt habe, habe ich mir mit Bleistift Markierungen an den Schraubstellen gemacht. Für die Schrauben habe ich etwas kleiner Löcher als den Durchmesser der Schrauben gebohrt und dann mit den Schrauben Gewinde ins Holz geschnitten.

Das dritte Servo wird gelegt eingebaut, weil es mit seinen zwei Beinchen die Ameise hin- und her-kippen soll. Die mittleren Beinchen sollten sich in der Mitte der Grundplatte bewegen. Die Position des mittleren Servos habe ich mir auf dem Brettchen wieder mit Bleistift angezeichnet. An die Seiten des Servos habe ich kleine Holzstückchen geleimt. Diese Holzstückchen habe ich von einer erbeuteten, ausgebrannten Silvesterrakete und mir zurecht gesägt. Ich bin mir fast sicher, ihr findet auch etwas passendes, ohne euren Etat übermäßig zu strapazieren. Das dritte, mittlere, liegende Servo habe ich dann nur noch mit zwei dünne Holzschrauben angeschraubt[14]. Auch hier ist Vorbohren ratsam.

14 Der von Ephraim Kishon beschrieben Monoschraubismus hat sich auch hier nicht bewährt.

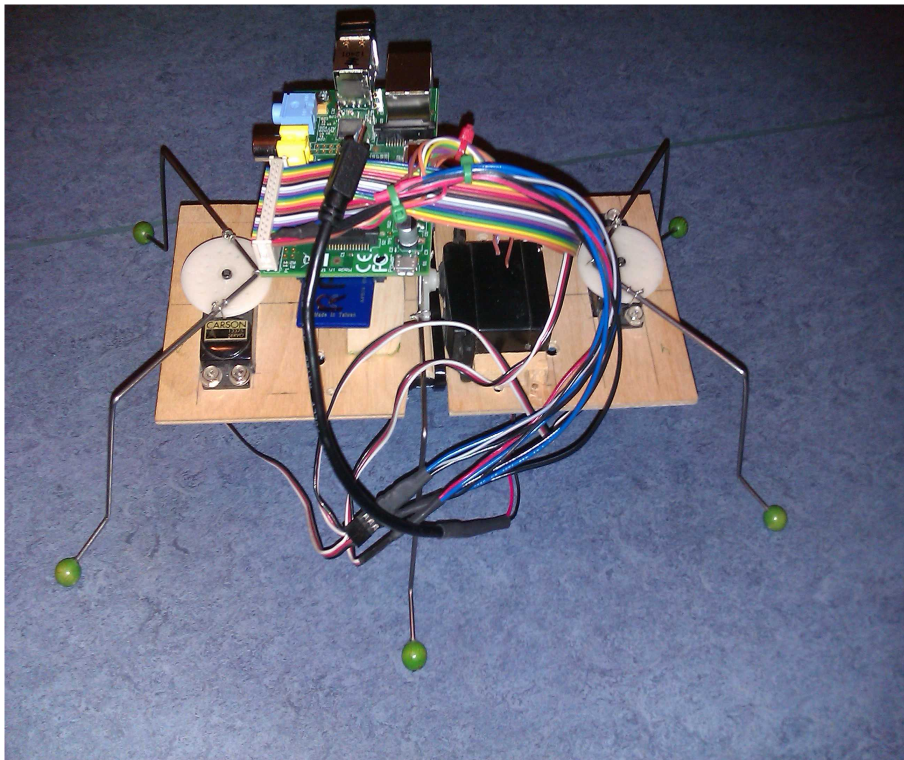


Abbildung 2: Die mit einfachen Mitteln zusammen gebaute Roboter-Ameise

Details vom Einbau des vorderen und mittleren Servos auf der Grundplatte seht ihr in Bild 3. Man kann auch deutlich die beiden fest geleimten Holzklötzchen rechts und links vom mittleren Servo sehen, die dieses fixieren.

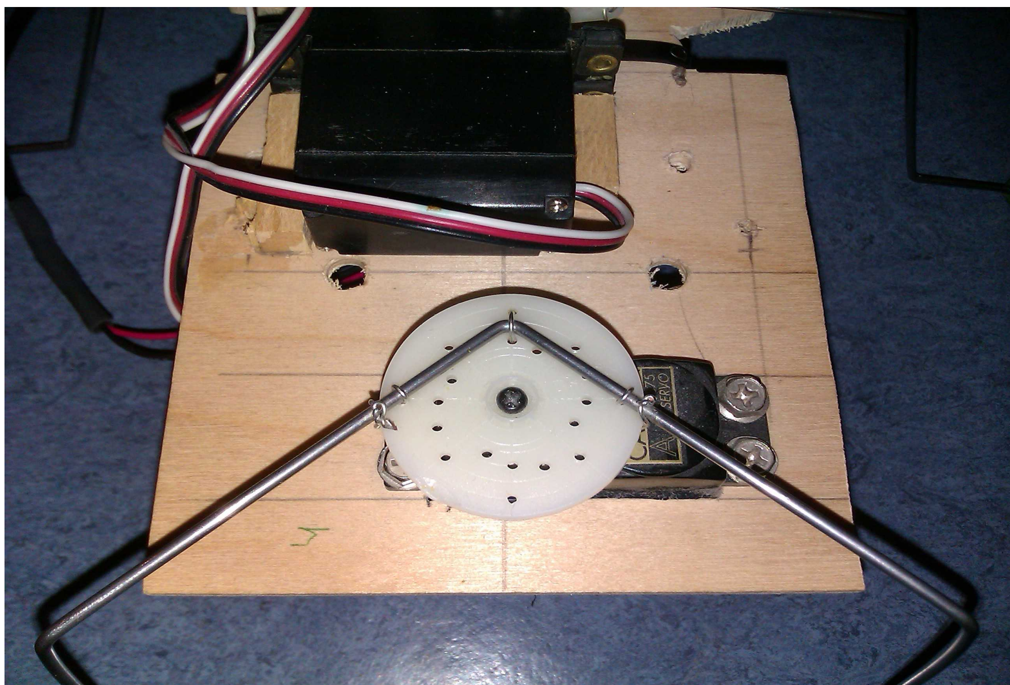


Abbildung 3: Details vom Einbau des vorderen und mittleren Servos

Die Position des hinteren und mittleren Servo auf der Grundplatte seht ihr in Bild 4 . Die beiden Holzsäulen nach oben verwende ich als Halterung für die Platine des Raspberry Pi.

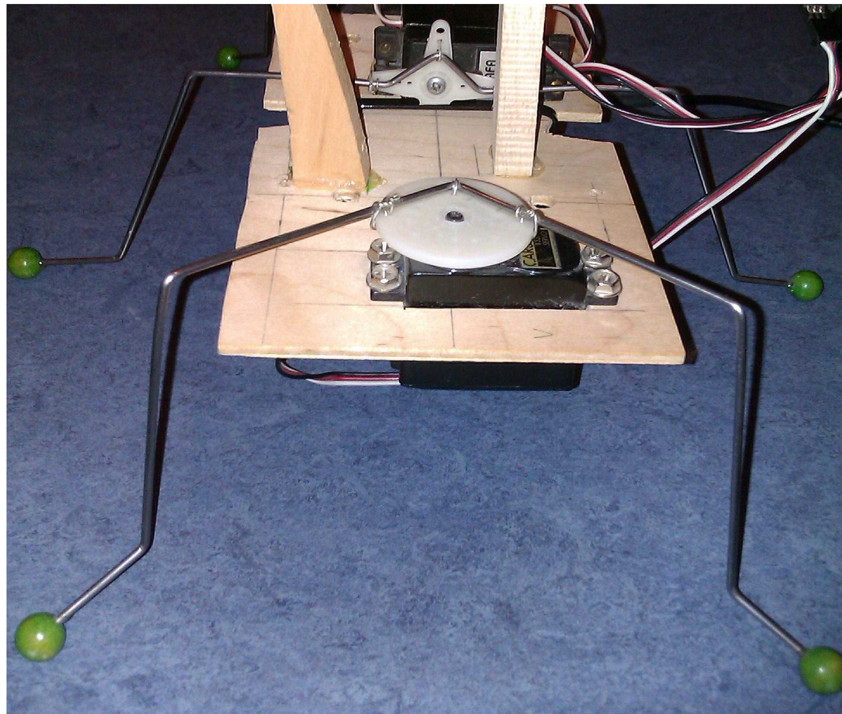


Abbildung 4: Hinteres und mittleres Servo auf der Grundplatte

2.2 Die Stromversorgung

Auf der Rückseite der Grundplatte habe ich den Batteriehalter für die Stromversorgung montiert. Den Bauch der Roboter-Ameise seht ihr in Bild 5. Ich habe NiMH Akkus mit geringer Selbstentladung vom Typ AA eingebaut.

Ihr solltet auf gar keinen Fall Alkali- oder andere 1,5 V Batterien einsetzen, da man mit vier 1,5 V Batterien auf 6 V kommt und damit über die zulässige maximale Betriebsspannung des Raspberry Pis von 5,5 Volt. Es drohen irreparable Schäden.

Mit einer maximalen Spannung von 1,25 V pro Zelle der NiMH-Akkus kommt man genau auf 5 V und bevor der Akku vollständig leer wird sinkt die Gesamtspannung unter 4,5 V und der Raspberry Pi Computer steigt langsam aus. NiMH Akkus mit geringer Selbstentladung haben eine etwas geringere Kapazität[15] als Standard NiMH Akkus und sind außerdem merklich teurer. Es lohnt sich aber hier etwas mehr zu investieren. Von NiCd-Akkus rate ich aufgrund des Memory-Effekts[16] und

15 Unter der Kapazität eines Akkus versteht man die Lademenge gemessen in mAh (Milliamperestunden). Meine Akkus haben 2500 mAh. Das bedeutet, das sie frisch aufgeladen etwa 2,5 Stunden lang 2 Stunden und 30 Minuten eine Strom von 1 A = 1000 mA liefern können, bevor sie leer sind. Je älter Akkus werden und je öfter sie aufgeladen wurden, um so geringer wird ihre Kapazität.

16 Unter dem Memory-Effekt bei NiCd-Akkus versteht man die extreme Reduzierung der Ladekapazität, wenn der Akku nicht vor dem Wiederaufladen nicht vollständig entleert ist. Moderne Ladegeräte haben deshalb eine Entladetaste oder einen Auswahlschalter für NiCd-Akkus. Bei NiMH-Akkus sollte man aber auf eine Entladung vor

der Umweltbelastung durch Cadmium generell ab, obwohl NiCd-Akkus auch eine Nennspannung von 1,2 V haben und den Raspberry Pi nicht zerstören können.

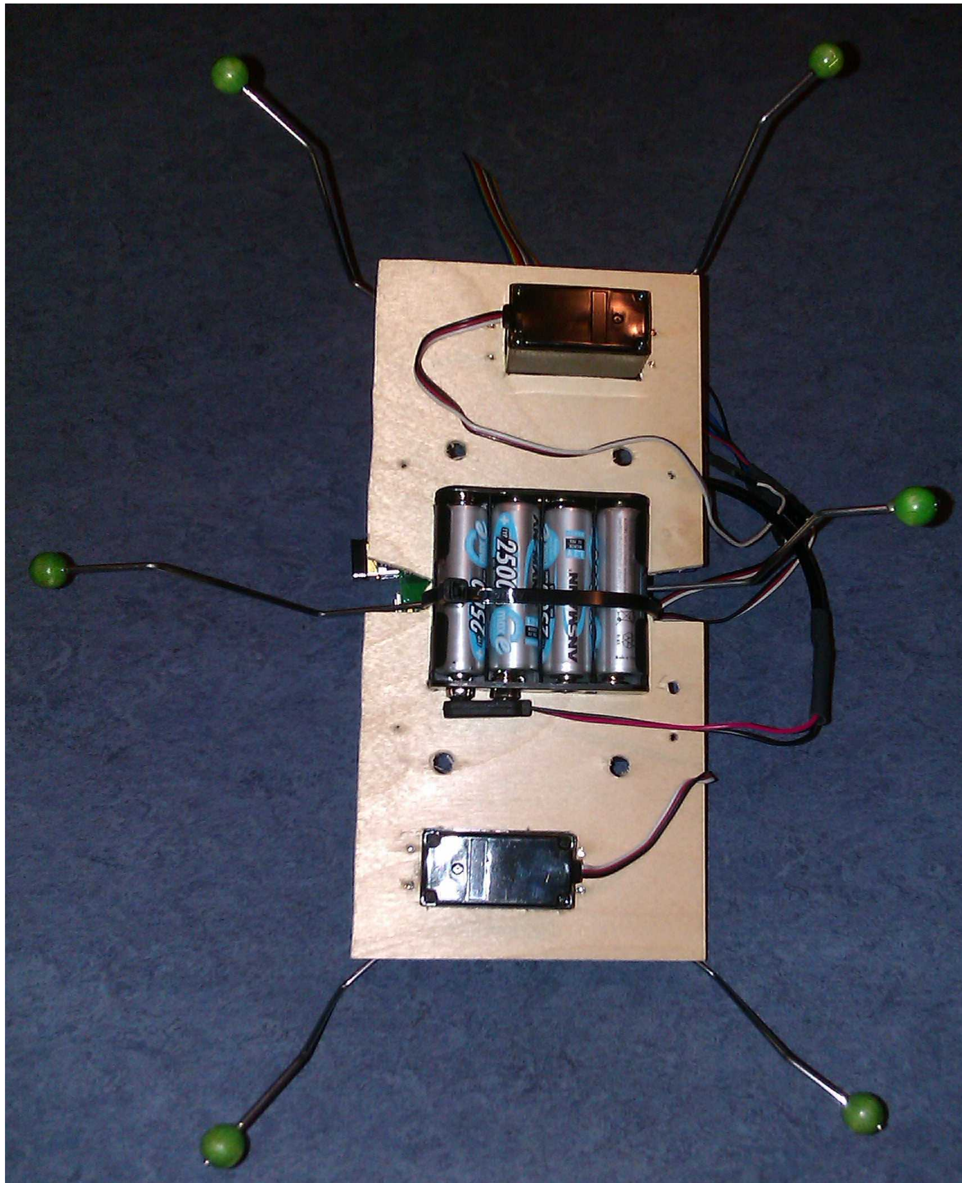


Abbildung 5: Die Rückseite der Roboter-Ameise mit dem Batteriehalter und den NiMH-Akkus

Den Batteriehalter habe ich mit einem Kabelbinder an der Grundplatte befestigt, was aber nicht so praktisch ist. Ihr solltet in festschrauben oder festkleben. In Bild 5 erkennt man außerdem das Anschlusskabel mit dem Batterieklipp, der als Steckkontakt für den + und – Pol des Batteriehalters dient. So einen Batteriehalter für 4 AA Akkus und auch den Batterieklipp mit Anschlusskabel bekommt ihr im Elektronikhandel um die Ecke oder online. Jetzt müsst ihr nur noch die Anschlusskabel mit einem Micro-B USB-Stecker verbinden, um den Raspberry Pi mit Strom zu versorgen. So einen Micro-B USB-Stecker kann man auch einzeln kaufen, billiger wird es aber, wenn ihr ein USB-Kabel mit Micro-B USB-Stecker kauft, diese durchschneidet und das andere

---

dem Laden verzichten. Erstens ist es nicht notwendig und zweitens reduziert man die Lebensdauer.

Ende weg schmeißt[17]. Zum Durchschneiden nehmt ihr einen Seitenschneider, aber ein starke Schere tut es auch.

Jetzt braucht ihr ein kleines scharfes Küchenmesser und ritzt etwa 2 cm vom abgeschnittenen Ende rings herum eine Kerbe in den Isoliermantel des Kabel. Aber Vorsicht, nicht zu tief in das Kabel schneiden, denn die Inneren Leitungen sollten nicht verletzt werden. In einem USB-Kabel sind vier Leitungen mit den Farben grün, weiß rot und schwarz. In die Isolierung dieser Leitungen solltet ihr nicht schneiden. Die äußere Isolierung kann man jetzt abziehen. Wenn das nicht so leicht geht, hilft häufig, wenn man das Kabel an der geritzten Stelle um 90° hin und her knickt. Wenn alles nichts hilft, muss man mit dem Messer noch mal vorsichtig nachritzen. Das ganze sieht dann aus wie in Bild 6.



Abbildung 6: Das abisolierte USB-Kabel

Die besseren USB-Kabel haben ein Drahtgeflecht um die vier Leitungen und manchmal auch noch so eine Art Alufolie. Das könnt ihr alles mit der Schere vorsichtig mit weg schneiden. Die weiße und die grüne Leitung sind die Datenleitungen für das USB-Signal. Die brauchen wir hier auch nicht und schneiden sie weg. Die rote Leitung ist die +5 V Leitung der Stromversorgung und die schwarze Leitung ist die Bezugsmasse GND[18] oder 0 V.

Jetzt kommt der große Auftritt des Lötkolbens[19], des Schrumpfschlauchs[20] und des Lötzinns[21]. Den Schrumpfschlauch ziehen wir natürlich <u>vor</u> dem Zusammenlöten über die Leitungen und das Kabel. Später ist der Jammer dann immer groß. Bild 12 zeigt so ein Elektroniker Stillleben.

Wir müssen also nur die roten und schwarze Leitungen von unserem Micro-USB-Kabel und dem Batterieklipp miteinander verbinden. Dazu nehmen wir wieder unser kleines Küchenmesser und ritzen die einzelnen Leitungen etwa 2 mm vom Ende an und ziehen die Isolierung ab. Die metallischen Drähtchen verdrillen wir zwischen unseren Fingern.

---

17 Ach, ich oute mich. Ich bin Elektronik-Messy und hebe Alles auf. Beim nächsten Projekt könnte ich es ja vielleicht wieder verwenden.

18 Was GND bedeutet erkläre ich euch später, wenn wir eine LED ansteuern.

19 Wenn wir eine Temperatur einstellen können, wählt man etwa 250°C für bleihaltiges und 300°C für bleifreies Lot. Ein vorheriger Test ist meist unumgänglich.

20 Wir brauchen verschiedene Durchmesser für die einzelnen Leitungen und für das Kabel. Ohne Schrumpfschlauch geht es notfalls auch. Dann nehmen wir Isolierband oder notfalls tut es auch Tesafilm.

21 Ich nehme immer noch bleihaltiges Lot, weil es einfacher zu löten ist. Ich hoffe, ihr verzeiht mir.

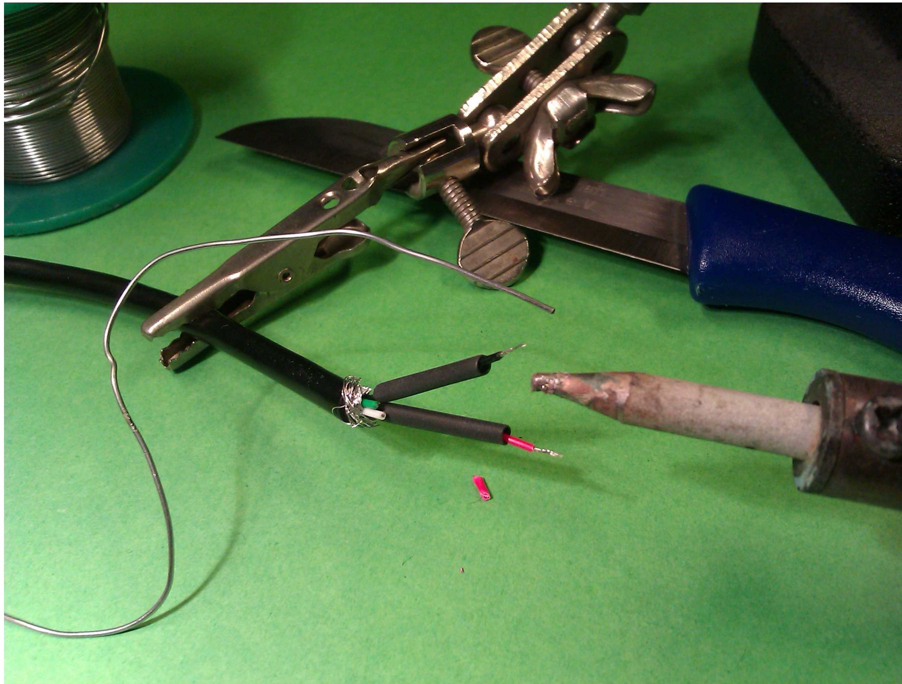


Abbildung 7: Das USB-Kabel vor dem Verlöten

Ihr werdet schnell feststellen, dass Menschen zum Löten anatomisch schlecht gebaut sind. Jedenfalls wünsche ich mir regelmäßig mindestens zwei Hände mehr. In Bild 7 seht ihr so eine Krokodilklemme, die an einen Ständer geschraubt ist. So etwas hole ich mir dann wenigstens zur Hilfe. Was ihr auf dem Bild noch gar nicht seht, ist das zweite Kabel mit den Leitungen vom Batterieklipp. Ihr werdet es irgendwie hinkriegen.

Wenn man mit Schrumpfschlauch arbeitet, kann man gegebenenfalls seinen Lötpfusch unter dem Schrumpfschlauch, den man über die Lötstelle zieht, verschwinden lassen. Um den Schrumpfschlauch schrumpfen zu lassen, muss man ihn erhitzen, dann zieht er sich zusammen. Das kann man mit einem Heißluftföhn oder einfach mit einem Feuerzeug machen. Dann sieht plötzlich alles richtig professionell aus wie in Bild .



Abbildung 8: Das zusammen gelötete Micro-USB Batterieklipp Kabel

2.3 Beine biegen

Als nächstes wollen wir unserer Roboter-Ameise die Beinchen montieren. Dazu besorgen wir uns einen Federstahl-Drahl von 2 mm Durchmesser und 1m Länge. Zuerst biegen wir die Vorder- und Hinter-Beinchen mit zwei starken Zangen, oder besser einem Schraubstock.

3 Erste Begegnung mit dem LINUX-Rechner Raspberry Pi

Nachdem wir zu Anfang etwas handwerklich gebastelt haben, werden wir uns jetzt mit der Computerei beschäftigen. Ich bin ein Fan des Betriebssystems LINUX. Die Anfänge von LINUX waren in Finnland, als der kleine Linus auf dem Schoß seines Großvaters die Tastatur eines Computers entdeckte[i]. Inzwischen ist LINUX ein Betriebssystem für die größten und kleinsten Computer. Und da so viele Leute an LINUX gearbeitet haben und noch immer arbeiten, ist es auch inzwischen richtig gut und dazu auch noch vollkommen kostenlos. Die meisten PCs haben Windows von Microsoft als Betriebssystem, oder laufen mit Mac OS von Apple. Wie diese Betriebssysteme funktionieren wissen die Hersteller. Dem Benutzer wird nur so viel mitgeteilt, wie er zum Arbeiten braucht. Will man mehr wissen, gibt es für den privaten Bastler häufig unüberwindbare Hürden. LINUX hat den Vorteil, dass das Betriebssystem für jeden, der es wissen möchte, offen ist. Alle Information zu LINUX sind frei zugänglich, wenn es auch immer schwerer für den Anfänger wird, sich hier zurecht zu finden. Aber es gibt gute Literatur zur Einführung[ii] und in diesem Buch versuche ich die Grundlagen anzusprechen, wenn sie benötigt werden.

Für den LINUX Anfänger ist zunächst etwas verwirrend, das LINUX nicht gleich LINUX ist. Es gibt unterschiedliche Distributionen[22]. Zu den bekanntesten zählen Fedorra, OpenSuse, Ubuntu und Debian. Während Fedorra und OpenSuse Abkömmlinge der Firmen RedHat bzw. Suse sind, ist Debian eine Distribution die sehr unabhängig ist und Ubuntu von der Firma Canonical hat sich davon abgeleitet. Das Betriebssystem für den Raspberry Pi ist Raspbian, das sich wie Ubuntu von Debian ableitet. Als ich dieses Buch schrieb hieß die Raspbian Version „wheezy“ und kann bei www.raspberrypi.org/downloads[23] kostenlos herunter geladen werden. Die Datei hat einen Namen, der ungefähr so heißt: **2012-12-16-wheezy-raspbian.zip**. Das Datum am Anfang heißt inzwischen bestimmt anders, aber ihr wisst, was ich meine, oder? Ihr ladet natürlich die gerade aktuelle Version herunter, denn mit jeder neuen Version sollte das Raspbian besser werden.

Jetzt müsst ihr euer Raspbian Betriebssystem noch auf die „Festplatte“ des kleinen LINUX-Rechners Raspberry Pi laden. Wie schon in der Einleitung erwähnt, hat der Raspberry Pi als Festplatte eine SD-Karte. Diese sollte 4 GB oder größer sein. Wir müssen also die herunter geladene Datei auf die SD-Karte übertragen. Dazu muss sie aber erst mal entpackt werden, denn sie kommt zunächst komprimiert[24] als zip-Datei. Der PC mit dem ihr Raspian aus dem Internet herunter geladen habt, ist entweder ein Windows-, Apple- oder LINUX-Rechner. Für LINUX-Rechner habe ich den Vorgang sehr ausführlich beschrieben.

---

22 Unter einer LINUX Distribution versteh man eine Variante von LINUX, die von einem bestimmten Herausgeber veröffentlicht wird.

23 Es ist nicht immer leicht sich auf Englischsprachigen Webseiten zurecht zu finden, wenn die Englischkenntnisse erst noch im aufgbau sind. Ich lasse mir dann durch die Webseite dict.leo.org helfen. LEO ist die Abkürzung von Link Everything Online und wurde von Leuten der TU-München ins Leben gerufen, und die verstehen was von Technik.

24 Komprimiert heißt hier, das die Datenmenge mit einem speziellem Verfahren verringert wurde. Entpacken bedeutet, das die ursprünglichen Daten wieder hergestellt werden.

3.1 Das Betriebssystem Raspbian auf eine SD-Karte laden mit Hilfe eines LINUX-PC

Wenn auf eurem PC, mit dem ihr Raspbian aus dem Internet geladen habt, ein LINUX Betiebssystem ist (egal welche Distribution) habt ihr Glück, denn mit LINUX geht es am schnellsten. Ich benutze in allen Betriebssystemen den Firefox Browser und der schlägt meistens schon vor, mit welchem Programm die Datei entpackt werden kann. Die entpackte Datei hat einen Namen, der bei mir so heißt: **2012-12-16-wheezy-raspbian.img**. Dabei handelt es sich um ein sogenanntes Image. Ein Image ist eine Eins-zu-Eins-Kopie der Daten eines Datenträgers. Es werden also auch die Partitionen[25] und Dateisysteme[26] mit kopiert. Um ein Image auf einen Datenträger zu übertragen gibt es für LINUX und Mac das Programm dd (disk dump). Dieses Programm ist sehr mächtig und man kann sich damit sogar seinen Computer zerstören. Wenn man nicht weiß was man tut, könnte man ungewollt die Festplatte des PCs überschreiben. Deshalb ist es ratsam erst mal nach zu schauen, wie die Datenträger und Laufwerke eigentlich heißen. Das finden wir mit dem LINUX-Programm[27] fdisk heraus. Öffnen wir dazu ein Konsolenfenster[28] indem wir Alt+F2 drücken und das Programm xterm aufrufen. Es meldet sich die bash[29]. Dort geben wir fdisk mit der Option -l[30] ein. Bei meinem LINUX kann man fdisk nicht so einfach aufrufen, ohne das man weiß wo es ist. Das lässt sich aber schnell mit dem Befehl cnf fdisk herausfinden:

```
axel@LINUX:/> cnf fdisk

Das Programm 'fdisk' ist verfügbar im Paket 'util-LINUX', das auf
ihrem System installiert ist.

Der absoluter Pfad für 'fdisk' ist '/usr/sbin/fdisk', daher ist es
wohl beabsichtigt, dass dieses nur von einem Benutzer mit
Superuser Rechten gestartet werden kann (z.B. root).
```

Wir brauchen also root Rechte um fdisk in dem Verzeichnis /usr/sbin auszuführen und besorgen uns diese mit dem vorangestellten Befehl sudo[31]. Bei mir sieht das dann so aus:

```
axel@LINUX:/> sudo /usr/sbin/fdisk -l
```

---

25 Partitionen sind Bereiche auf einem Datenträger. Bei Windows werden sie häufig mit Buchstaben und einem Doppelpunkt gekennzeichnet, z.B. C: für die erste Partition der Festplatte (A: und B: wurden historisch für Floppylaufwerke verwendet, aber die kennt heute kaum noch jemand). Bei LINUX heißen die Partitionen z.B. sda1 oder sda2.

26 Dateisysteme beschreiben wie die Dateien auf dem Datenträger abgespeichert werden. Das bekannteste Dateisystem ist FAT16 oder FAT32 (file allocation table mit 16 oder 32 bit). FAT kann von Windows, Mac und LINUX gelesen und geschrieben werden. Modernere Dateisysteme sind z.B. ext4 oder NTFS. NTFS gehört aber Microsoft und kann von LINUX zwar gelesen, nicht aber beschrieben werden. Bei ext4 verweigert Windows die Zusammenarbeit.

27 Unter Windows gibt es auch ein Programm fdisk. Das arbeitet ähnlich, aber nicht ganz genau so.

28 Konsolen bestanden früher, als es noch keine grafische Benutzeroberfläche gab, aus einem Bildschirm und einer Tastatur, mit der man die Kommandos an den Computer gab. Die Maus war noch nicht erfunden. In einem Konsolenfenster kann man diese Art der Computerei nachempfinden.

29 Bash steht für bourne-again shell. Bourne-again weil die shell eigentlich schon tot war. Mit einer shell kann man Befehle an das Betriebssystem senden oder Programme aufrufen.

30 Häufig können LINUX Kommandos und Programme (der Unterschied zwischen Kommando und Programm ist nicht so ganz abgegrenzt) mit Optionen, die mit einem Minuszeichen und einem Buchstaben an das Programm übergeben werden, beeinflusst werden. Hier steht das -l für „list“ (auflisten).

31 Der Befehl sudo steht für „superuser do“ (Der Superuser, also root, führt den Befehl aus).

```
root's password:

Disk /dev/sda: 60.0 GB, 60022480896 bytes
255 Köpfe, 63 Sektoren/Spur, 7297 Zylinder, zusammen 117231408
Sektoren
Einheiten = Sektoren von 1 × 512 = 512 Bytes
Sector size (logical/physical): 512 bytes / 512 bytes
I/O size (minimum/optimal): 512 bytes / 512 bytes
Disk identifier: 0x218436fa

   Gerät  boot.     Anfang        Ende     Blöcke   Id  System
/dev/sda1   *         2048   116198144    58098048+  83  LINUX
/dev/sda2        116198152   117226304      514076+  82  LINUX Swap

Disk /dev/sdb: 500.1 GB, 500107862016 bytes
255 Köpfe, 63 Sektoren/Spur, 60801 Zylinder, zusammen 976773168
Sektoren
Einheiten = Sektoren von 1 × 512 = 512 Bytes
Sector size (logical/physical): 512 bytes / 512 bytes
I/O size (minimum/optimal): 512 bytes / 512 bytes
Disk identifier: 0x000ccfb6

   Gerät  boot.     Anfang        Ende     Blöcke   Id  System
/dev/sdb1               63     4209029     2104483+  82  LINUX Swap
/dev/sdb2   *      4209030    86130687    40960829   83  LINUX
/dev/sdb3        128075776   947277823   409601024   83  LINUX
```

Immer wenn wir Befehle oder Programme mit voran gestelltem sudo ausführen, werden wir erst mal nach dem Passwort gefragt. Nachdem wir das Passwort für root eingegeben haben kommen jede Menge Informationen. Wie ihr seht, habe ich zwei Laufwerke. sda mit 60 GB (das ist meine SSD[32]) und sdb mit 500 GB (das ist meine Festplatte[33]). Auf sda gibt es zwei Partitionen. Auf der Partition sda1 habe ich mein LINUX Betriebssystem und die Programme. Die zweite Partition sda2 benötigt LINUX intern. Meine Daten habe ich auf der Festplatte sdb. Auf sdb gibt es 3 Partitionen.

Die Laufwerke sda und sdb sind also für dd unbedingt tabu, sonst ist alles hin. Um heraus zu bekommen, wie unsere SD-Karte heißt, legen wir diese jetzt in den SD-Karten Leser des LINUX-PCs und geben den Befehl erneut ein:

```
axel@LINUX:/> sudo /usr/sbin/fdisk -l
root's password:

Disk /dev/sda: 60.0 GB, 60022480896 bytes
```

32 SSD steht für „solid state disk“. Eine SSD ist schneller als ein Festplattenlaufwerk und so booted mein Rechner schneller.

33 Der Name Festplatte stammt noch aus der Zeit, als viel mit Floppylaufwerken gearbeitet wurde. Floppy heißt eigentlich felxibel und damit meinte man eine runde flexieble Magnetscheibe, die in so einer Art quadratischem Briefumschlag mit Löchern steckte. Im Gegensatz zu Floppies, die man tauschen konnte wie heute CDs, bestehen Festplatten aus mehreren starren übereinander angeordneten Magnetscheiben, die sich in staubfrei abgedichteten Gehäusen drehen.

```
255 Köpfe, 63 Sektoren/Spur, 7297 Zylinder, zusammen 117231408
Sektoren
Einheiten = Sektoren von 1 × 512 = 512 Bytes
Sector size (logical/physical): 512 bytes / 512 bytes
I/O size (minimum/optimal): 512 bytes / 512 bytes
Disk identifier: 0x218436fa

   Gerät  boot.     Anfang        Ende     Blöcke   Id  System
/dev/sda1   *         2048   116198144   58098048+  83  LINUX
/dev/sda2       116198152   117226304     514076+  82  LINUX Swap
.
.
.
.
Disk /dev/sde: 16.1 GB, 16130244608 bytes
64 Köpfe, 32 Sektoren/Spur, 15383 Zylinder, zusammen 31504384
Sektoren
Einheiten = Sektoren von 1 × 512 = 512 Bytes
Sector size (logical/physical): 512 bytes / 512 bytes
I/O size (minimum/optimal): 512 bytes / 512 bytes
Disk identifier: 0x000f06a6

   Gerät  boot.     Anfang        Ende     Blöcke   Id  System
/dev/sde1             8192      122879      57344    c  W95 FAT32
/dev/sde2           122880    31504383   15690752   83  LINUX
```

Nach erneuter Eingabe des Passworts kommt wieder der gleiche Text aber nicht ganz. Diesmal ist die Ausgabe mit Angaben zum Laufwerk[34] sde erweitert. Meine SD-Karte ist also unter /dev/sde zu finden. Wir erkennen außerdem, dass auf dieser SD-Karte schon zwei Partitionen existieren auf denen bereits Daten sind[35]. Diese Daten werden jetzt überschrieben und sind dann weg. Falls ich sie noch brauche, müsste ich sie jetzt sichern.

Übrigens könnte ich statt fdisk -l auch den Befehl df -h benutzen. Dieser ist aber nicht immer zuverlässig, da die SD-Karte erst erkannt wird, wenn auf sie schon mal zugegriffen wurde.

Um unsere Image-Datei mit dem Namen **2012-12-16-wheezy-raspbian.img** (bei euch so ähnlich) auf die SD-Karte zu schreiben, wechseln wir mit cd[36] nun in das Verzeichnis, in der sich die Image-Datei befindet. Dazu geben wir in das Konsolenfenster den Befehl:

```
axel@LINUX:~> cd /home/axel/Downloads
axel@LINUX:~/Downloads>
```

Hier schauen wir mit ls[37] nach, ob die Datei auch da ist:

```
axel@LINUX:~/Downloads> ls *wheezy*
```

34 Die Dinger heißen immer noch Laufwerk, obwohl sich bei SSDs und SD-Karten nichts bewegt.
35 Ehrlich gesagt: Ich hab einfach meine SD-Karte genommen, auf der schon Raspbian ist.
36 Der Befehl cd steht für „change directory“ (wechsele das Verzeichnis).
37 Der Befehl ls steht für „list“ und listet die Verzeichnisse und Dateien im aktuellen Verzeichnis auf.

```
2012-12-16-wheezy-raspbian.img
2012-12-16-wheezy-raspbian.zip
```

Das ist offensichtlich der Fall und wir können das Image auf der SD-Karte jetzt mit dem Befehl:

```
dd bs=1M if=2012-12-16-wheezy-raspbian.img of=/dev/sde
```

Ihr müsst hier natürlich statt /dev/sde natürlich euren Pfad zur SD-Karte angeben. Mit der Option bs=1M haben wir dd mitgeteilt, dass 1Mbit Blöcke übertragen werden sollen. Der Befehl dd braucht lange (das kann schon mal mehrere Minuten dauern) und sollte nicht unterbrochen werden. Wenn alles gut gegangen ist, meldet sich nach einiger Zeit der Prompt[38] zurück. Unser Raspbian Betriebssystem ist nun auf der SD-Karte und wir können diese jetzt in den Raspberry Pi stecken.

3.2 Das Betriebssystem Raspbian auf eine SD-Karte laden mit Hilfe eines Windows-PC

Für Windows-Rechner benötigen wir die Programme WinZip und Win32DiskImager. Mit WinZip entpacken wir die herunter geladene Datei, so dass wir die Datei **2012-12-16-wheezy-raspbian.img** vorliegen haben. Dann stecken wir die SD-Karte in den Kartenleser und schauen uns an welchen Laufwerksbuchstaben die SD-Karte hat.

Um an den Win32DiskImager heran zu kommen gehen wir auf die Webseite http://sourceforge.net/projects/win32diskimager/ , starten den Download und installieren das kostenlose Programm. Im Win32DiskImager geben wir an, wo die Image-Datei **2012-12-16-wheezy-raspbian.img** zu finden ist. In der Device Box wählen wir den Laufwerksbuchstaben der SD-Karte aus. Vorsicht, wenn ihr den falschen Buchstaben erwischt, lauft ihr Gefahr euren PC zu schrotten. Jetzt müsst ihr aus „write“ klicken und abwarten bis der Schreibvorgang zu Ende ist. Das kann einige Zeit dauern.

3.3 Das Betriebssystem Raspbian auf eine SD-Karte laden mit Hilfe eines Apple-PC

Das Apple Mac OS X Betriebssystem gehört wie LINUX zu den Unix-artigen Betriebssystemen. Daher läuft vieles ähnlich ab wie bei LINUX-PCs. Ihr öffnet ein Terminal und wechselt auf die Komadozeile. Mit dem Befehl df -h schaut ihr euch an, wie die Laufwerke heißen. Jetzt legt ihr die SD-Karte ein und gebt wieder df -h ein. Jetzt solltet ihr wissen, wie eure SD-Karte heißt (z.B. /dev/disk3s1). Die SD-Karte muss jetzt mit dem Befehl:

```
diskutil unmount /dev/disk3s1
```

zum Überschreiben vorbereitet werden. Das Laufwerk wird nun umbenannt mit:

```
/dev/disk3s1 --> /dev/rdisk3
```

Mit dem Befehl dd wird wie bei LINUX das Image auf die SD-Karte übertragen. Dazu brauchen wir Superuser-Rechte mit sudo:

38 Der Prompt ist der Text am Anfang jeder Zeile der bash, in die ein Befehl eingegeben werden kann. Bei mir ist das: axel@LINUX:/>

```
sudo dd bs=1m if=<pfad>[39]/2012-12-16-wheezy-raspbian.img
of=/dev/rdisk3
```

Der Befehl dd braucht einige Zeit und muss beendet sein, bevor mit dem Kommando:

```
diskutil eject /dev/rdisk3
```

die SD-Karte freigegeben werden kann.

4 Den Raspberry Pi zum ersten mal booten[40]

Jetzt können wir den Raspberry Pi zum ersten mal in Betrieb nehmen. Dazu schließen wir den Monitor über ein HDMI-Kabel an und die Tastatur und Maus über die beiden USB-Anschlüsse. Falls wir keinen Monitor mit HDMI-Anschluss haben, tut es auch einer mit DVI-Anschluss. Dazu brauchen wir dann aber ein HDMI-DVI Kabel oder einen HDMI-DVI Adapter[41]. Um den Raspberry Pi zu starten müssen wir jetzt nur noch ein Netzteil mit 5V Ausgangsspannung und Micro-USB Stecker anschließen. Meistens tut es hier ein Handy-Ladegerät. Der Raspberry Pi hat keinen Schalter und startet sofort. Die Leuchtdiode, an der PWR[42] steht, leuchtet jetzt rot. Die Leuchtdiode neben ACT[43] beginnt unregelmäßig grün zu blinken. Wir sollten jetzt auf dem Monitor lauter Meldungen vom booten sehen. Irgendwann müssen wir uns am Raspberry Pi anmelden. Standardmäßig ist als Benutzername: „pi“ einzugeben. Das Passwort lautet „raspberry“[44]. Nach einiger Zeit sehen wir auf dem Monitor folgendes Auswahl-Menü[45]:

---

39 <pfad> steht hier für den Pfad zur Image-Datei.

40 Unter booten versteht man das starten des Betriebssystems. Wörtlich bedeutet booten so viel, wie die Stiefel anziehen.

41 Leider unterstützt der Raspberry Pi kein VGA. HDMI zu VGA Adapter sind relativ teuer und nicht zu empfehlen. Man kann aber den Videoausgang auf der gelben Chinch Buchse verwenden und an einen Fernseher anschießen.

42 PWR steht für englisch „power“, was auf deutsch „Leistung“ heißt.

43 ACT steht für englisch „acknowledge“, was auf deutsch so viel wie „Arbeitet“ heißt.

44 Achtung! Beim ersten Start steht die Tastatureinstellung auf Englische Tastatur. Dadurch sind die Tasten „Z“ und „Y“ vertauscht. Gebt also beim ersten mal als Passwort nicht „raspberry“ sonder „raspberrz“ ein, wenn ihr eine Deutesche Tastatur angeschlossen habt.

45 Wir können dieses Menü jederzeit wieder starten, wenn wir auf der Komandozeile den Befehl „sudo raspi-config“ eingeben.



Abbildung 9: Das Konfigurations-Menü des Raspberry Pi

Gehen wir die Auswahlmöglichkeiten im Detail mal durch:

4.1 info:

Zeigt einen kurzen Text an, der ungefähr übersetzt so lautet:

„Dieses Werkzeug stellt einen direkten Weg zur Verfügung um erste Einstellungen des Raspberry Pi vorzunehmen. Obwohl es jederzeit gestartet werden kann, gibt es mit einigen Optionen Schwierigkeiten, wenn ihr sie zu stark verändert.“

4.2 expand_roots:

Mit dieser Option wird die root Partition automatisch angepasst. Das ursprüngliche Image hat etwa 2GB. Habt ihr das Image auf eine 4GB oder größere SD-Karte übertragen, so wird jetzt der Rest so konfiguriert, dass ihr beim nächsten Start die volle SD-Kartengröße zur Verfügung habt.

4.3 Overscan:

Hier können wir den Overscan ausschalten (disable) oder einschalten (enable). Ist der Overscan aktiviert, wird ein schwarzen Rand um das Bild gezeichnet. Das kann ganz nützlich sein, wenn wir nicht den HDMI-Anschluss nutzen, sondern den gelben, analogen Videoausgang, zum Beispiel einen alten Fernseher zu nutzen. Die Änderung der Einstellung wird erst beim nächsten Start des Pi übernommen. Eigentlich wird hier nur in der Datei /boot/config.txt der Parameter disable_overscan gesetzt. Das könnten wir aber auch mit einem Texteditor manuell eintragen.

4.4 configure-keyboard:

Hier könnt ihr eure Tastatur angeben. Zum Beispiel ob ihr einen separaten Nummerblock habt und ob eure Tastatur Deutsch ist. Ich habe 105 Tasten und German eingegeben. Die anderen Fragen habe ich alle mit „ok“ beantwortet.

Voreingestellt ist eine englische Tastatur mit englischem Layout[46]. Über den Punkt "Other" bekommen wir eine zusätzliche Auswahl, in der erst mal die Grundanordnung auswählt wird. Zu bemerken ist noch, dass es nicht einfach nur ein deutsches Layout gibt, sondern sogar zwischen der Schweiz und Österreich noch unterschieden werden kann.

Die Frage, ob per Control+Alt-Backspace, der X-Server beendet werden soll, beantworten wir mit "Yes". So haben wir die Möglichkeit die grafische Oberfläche mit dieser Tastenkombination zu beenden, um ein nicht mehr reagierendes Programm zu beenden.

Damit das System die Einstellungen übernimmt, braucht es eine Weile.

4.5 change_pass:

Hier kann das Passwort des Benutzers „pi" verändert werden. Will man das Passwort ändern, sollte man sich eines überlegen, das möglichst mit allen Tastatureinstellungen gleich eingeben werden kann. Dann erspart man sich die Überraschung mit dem vertauschten z und y.

4.6 change_locale:

Hier wählte ich „de_DE.UTF-8 UTF-8". Mit diesem Zeichensatz wählen wir die Sprache des Systems[47]. Zur Auswahl markieren wir den den Punkt mit der Leertaste. Mit der TAB-Taste kommen wir auf die Schaltfläche "Ok" und wählen wieder "de_DE.UTF-8" aus. Dadurch geben Programme deutsche Texte aus, wenn diese verfügbar sind. Wieder benötigt das System einen Augenblick um die Einstellung zu übernehmen.

4.7 change_timezone:

Hier teilen wir dem Raspberry Pi mit, in welcher Zeitzone wir uns befinden. Ich ahbe „Europe" und „Berlin" ausgewählt.

4.8 memory_split:

Der Hauptspeicher, das ist der Chip[48], der auf den Prozessor gelötet ist, hat eine Größe von 512 MB[49]. Diesen Speicher teilen sich Hauptprozessor und Grafikprozessor[50]. Beide haben Zugriff auf einen bestimmten Anteil des RAM[51]. Es gibt vier Einstellmöglichkeiten für den Grafikprozessor: 32\64\128\256 MB. Den Rest bekommt der Hauptprozessor. Wenn ihr keine aufwändigen Videofunktionen benötigt, reichen 32 MB. Für 3D-intensive Anwendungen wählt ihr 128 MB. Die Standard-Einstellung[52] ist 64MB. Auch diese Einstellung wird erst nach dem Neustart wirksam.

4.9 overclock[53]:

Hier kann man die Taktfrequenz, mit der der Prozessor seine Befehle abarbeitet, einstellen.

46 „Layout" ist natürlich englisch und bedeutet hier die „Anordnung" der Tasten.
47 Mit „System" ist hier das Betriebssystem des Raspberry Pi gemeint, also hier Raspbian.
48 „Chip" ist natürlich wieder englisch und bedeutet soviel wie „elektronischer Baustein".
49 Die älteren Versionen haben teilweise nur 256MB.
50 Der Graphikprozessor stellt das Bild auf dem Monitor zusammen. Der Hauptprozessor macht alles andere.
51 „RAM" ist die Abkürzung für „random access memory", das ist ein Speicher auf den man genauso gut Daten abspeichern (schreiben), wie lesen kann. Das RAM ist nur ein anderer Name für den Hauptspeicher.
52 Für Standard-Einstellung wird im englisch der Begriff „default" verwendet.
53 „overclock" heißt hier nicht über der Uhr sondern „übertakten".

Standardmäßig arbeitet der Prozessor mit 700 MHz. Dynamisches übertakten bedeutet, das die höhere Taktrate nur bei Bedarf verwendet wird. Dadurch lässt sich Energie sparen. Das ist zwar bei der Stromrechnung zu vernachlässigen, aber wir müssen bedenken, das die Energie im Prozessor vollständig in Wärme umgewandelt wird und dieser dadurch überhitzen kann. Zwar kontrolliert sich der Prozessor selbst und taktet langsamer, wenn er wärmer als 85°C wird, aber eine andauernde hohe Temperatur kann die Lebensdauer des Prozessors verkürzen. Bei einer zu hohen Taktrate, kann sich der Prozessor auch mal verschlucken und das System wird instabil. Dann sollte man wieder zu geringeren Taktraten zurück kehren. Ich habe hier nichts verändert, denn der Vorteil durch die höhere Taktrate ist nicht sonderlich spürbar in unserem Ameisen Projekt.

4.10 ssh:

Mit ssh haben wir die Möglichkeit über das Netzwerk die Kommandoebene des Raspberry Pi von entfernten Rechnern zu bedienen. Es gibt auch Optionen von ssh, die es erlauben Grafik-Fenster zu übertragen (zum Besispiel ein Kamera Bild). Davon werden wir im Laufe des Projekts ausgiebig Gebrauch machen und wir schalten die ssh-Fähigkeit ein (enable).

4.11 boot_behaviour:

Hier können wir einstellen, ob wir beim Start direkt auf die grafische LXDE[54]-Oberfläche wollen oder nicht. Wir können, falls wir hier „No“ eingegeben haben aber auch jederzeit auf der Kommandoebene „startx“ eingeben und wechseln dann im Nachhinein auf die grafische Benutzerberfläche.

4.12 update:

Dies ist der letzte Eintrag des Menüs. Wenn der Raspbery Pi eine Netzwerkverbindung zum Internet hat, versucht er hier das Tool[55] raspi-config zu aktualisieren.

Durch das drücken der Tabulator Taste kommen wir zu <Finish> und können damit das Menü beenden. Ich empfehle den Raspberry Pi jetzt neu zu starten. Dazu sollte man ihn ordentlich herunterfahren[56]. Dazu verwenden wir auf der Kommandozeile den Befehl „sudo shutdown now“. Auf der grafischen Benutzeroberfläche klicken rechts unten aud das kleine rote Kästchen mit der 1 im Ring und wählen dann Shutdown. Jetzt müssen wir etwas warten bis der Bildschirm keine Meldungen mehr liefert und die grüne ACT Leuchtdiode nicht mehr blinkt. Vorsichtshalber warten wir noch etwas bevor wir den USB-Stecker der Stromversorgung ziehen.

5 Mit LINUX etwas vertraut werden

Wenn wir den Raspberry Pi jetzt starten, also die Stromzufuhr am Micro-USB Stecker herstellen, sollten wir jetzt nach einiger Zeit das folgende Bild sehen. Ist das nicht der Fall, weil wir zum

---

54 Die Benutzeroberfläche, das ist der Teil des Betriebssystems der die Fensterverwaltung übernimmt, ist bei Raspbian LXDE. Diese Abkürzung steht für „Lightweight X11 Desktop Environment“. Es ist sozusagen Leichtgewicht unter den Benutzeroberflächen und benötigt wenig Specher und Rechenleistung und ist daher sehr gut für den Raspberry Pi geeignet.

55 „tool“ heißt auf englisch „Werkzeug“. Unser gestartetes Programm „raspi-config“, das uns dieses Menü zur Verfügung gestellt hat, wird auch als „tool“ bezeichnet.

56 Das Herunterfahren ist das Gegenstück zum Booten. Hier werden alle Programme beendet und die Prozesse ordentlich abgeschlossen.

Beispiel in „boot_behaviour“[57] eine andere Einstellung gewählt haben, sollten wir jetzt den Befehl „startx“ eingeben.

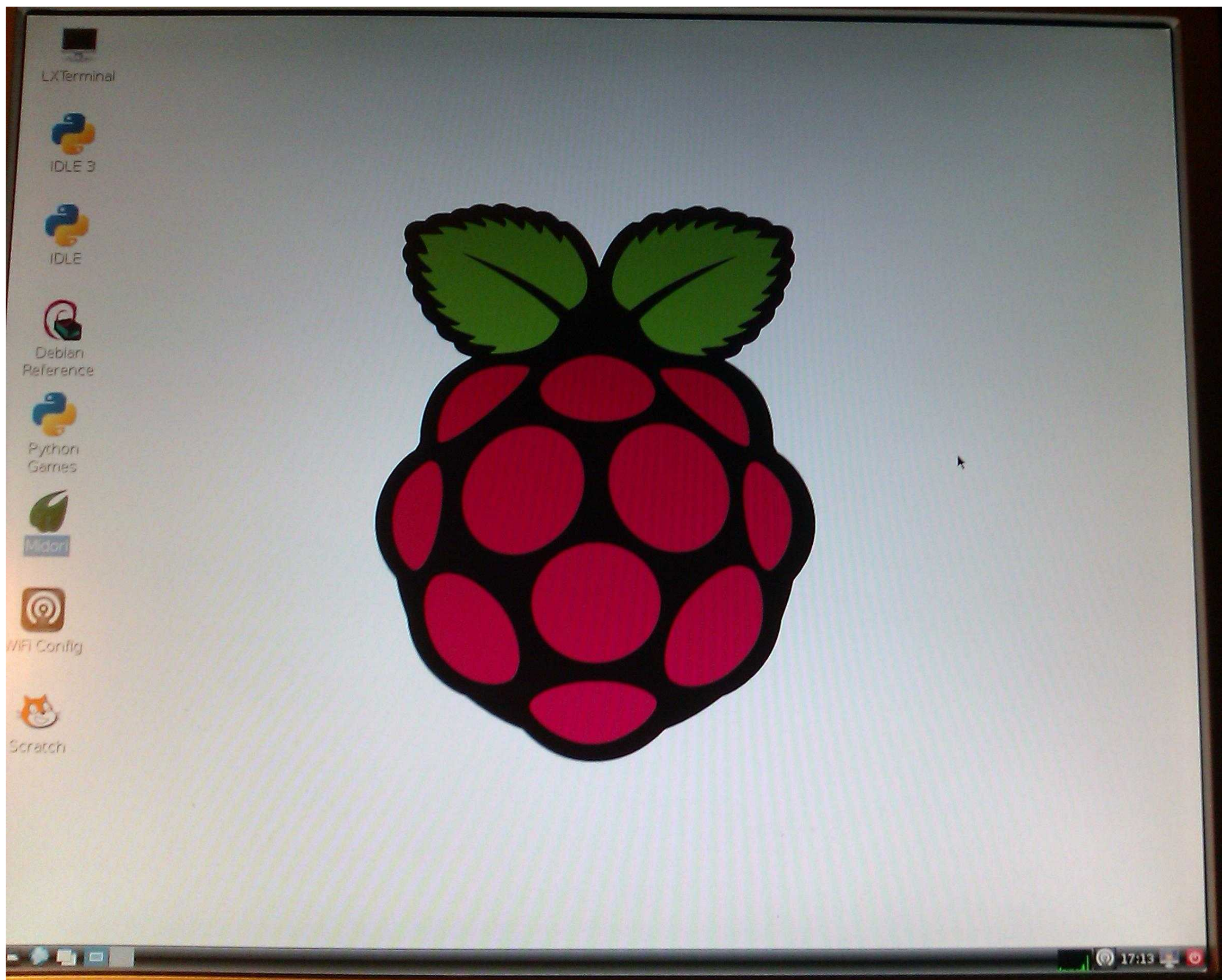


Abbildung 10: Montiorbild nach dem Starten des Raspberry Pi

Wir sehen jetzt die Benutzeroberfläche[58] von LXDE. LXDE wurde 2006 von dem Taiwaner Hong Yen Lee[59] ins Leben gerufen und erfreut sich nicht nur seit dem internationaler Beliebtheit und sondern hat auch die Unterstützung von vielen Entwicklern gefunden. Unten auf dem Desktop finden wir die Task-Leiste[60]. Klicken wir mit der linken Maustaste ganz links unten auf das LXDE Symbol öffnet sich ein Auswahl-Menü. Hier können wir über Unter-Menüs verschiedene Programme starten. Mit diesen Programmen kann jeder für sich ein bisschen rumspielen und sich vertraut machen. Rechts neben dem LXDE Symbol in der Task-Leiste ist das Symbol für den File-Manager[61]. Den öffnen wir mal mit der linken Maus-Taste. Es erscheint das Fenster des File-

57 Siehe Kapitel davor
58 Manchmal wird dies auch als Desktopsystem bezeichnet.
59 An dieser Stelle möchte ich mich bei Hong Yen Lee bedanken, denn LXDE ist richtig gut geworden.
60 Task-Leiste ist natürlich denglisch und heißt richtig „taskbar“ und wird auf deutsch mit „Startleiste“ übersetzt.
61 „File“ ist das englische Wort für Datei. Der File-Manager ist sowas wie bei Windows der Explorer.

Managers. Links in dem Kasten sehen wir jetzt ein Haus-Symbol neben dem steht „pi“[62]. Darunter ist das Symbol eines Bildschirms neben dem steht „Desktop“. Wenn wir hier klicken[63], sehen wir die gleichen Symbole, die auch links im großen Bildschirm zu sehen sind. Der Desktop[64] zeigt also das Zeug, was auf dem Schreibtisch liegt. Unter „Desktop“ ist ein Mülleimer-Symbol neben dem Rubbish steht. Wenn wir bei Rubbish klicken, sehen wir die Dateien, die wir gelöscht haben[65]. Unter Rubbish steht Applications[66]. Hier sehen wir die Programme, die wir teilweise auch über den Desktop oder das LXDE-Symbol links unten in der Task-Leiste erreichen können.

Übrigens haben alle Fenster, nicht nur das Fenster des File-Managers, bei LXDE immer den gleichen Rahmen. Ganz oben ist ein schwarzer Balken in dem eine nützliche Information steht. Rechts oben in dem schwarzen Balken gibt es drei Symbole. Klicken wir auf das Linke von den dreien, so verschwindet unser Fenster in der Task-Leiste. Ein Klick auf das Symbol in der Task-Leiste, und das Fenster ist wieder da. Klicken wir auf das Mittlere, nimmt unser Fenster den gesamten Desktop ein. Wenn wir nochmal darauf drücken, nimmt das Fenster wieder seine ursprüngliche Größe ein. Mit dem ganz Rechten Symbol, können wir das Fenster schließen. Das funktioniert wie bei vielen anderen Betriebssystemen auch.

Aber klicken wir erst mal auf das Haus-Symbol in unserem File-Manager, das neben dem „pi“ steht. Jetzt sollten wir oben in der Zeile so was wie „/home/pi/“ sehen und um zu verstehen warum das so ist, müssen wir uns etwas damit beschäftigen, wie das LINUX File-System aufgebaut ist. Erfreulicher weise ist das sehr logisch, so dass wir es, wenn wir es verstanden haben, was nicht schwer ist, auch leicht merken können.

5.1 LINUX Verzeichnis System

Die Verzeichnisstruktur bei LINUX ist wie ein Baum, mit Ästen. Unten ist die Wurzel[67] und dann verzweigt es sich. Bei „/home/pi“ sind wir also schon auf dem Haupt-Ast „home“ und dem Unter-Ast „pi“. Zwischen den Verzweigungen steht bei LINUX immer der Schrägstrich[68]. Wir können von unserem Ast „pi“ etwas „herunter klettern“ indem wir neben dem Haus-Symbol auf den kleinen grünen Kreis mit dem Pfeil nach oben klicken. Jetzt sind wir auf dem Haupt-Ast „/home“. Statt Ast werde ich ab jetzt immer von Verzeichnis[69] reden, aber wir stellen uns das ganze trotzdem wie ein Baum vor. In dem Verzeichnis „/home“ gibt es erst mal nur ein einziges Unterverzeichnis mit dem Namen pi. Statt Verzeichnis wird auch oft der Begriff „Ordner“ gebraucht und das Symbol[70] von „pi“ sieht auch so ein bisschen so aus, wie ein Ordner. Klicken wir jetzt noch mal auf den grünen runden Kreis mit dem Pfeil nach oben, sind wir an der Wurzel angekommen. Da erkennen wir daran, dass jetzt in der oberen Zeile nur noch „/“ steht. Vom Wurzelverzeichnis verzweigen sich viele Unter-Verzeichnisse. Und wir finden auch das Verzeichnis „home“ als eines unter vielen wieder.

Der File-Manager ist ein ziemlich mächtiges Programm, dass uns eine Menge Informationen bereit hält. Ihr probiert einfach alles aus, so macht man sich am schnellsten damit vertraut. Ich möchte euch nur ein paar wichtige Dinge zeigen, die wir später häufiger brauchen. Klickt mal oben in der

62 „pi“ steht da, weil das unser Benutzername ist; ihr erinnert euch.
63 Wenn ich „klicken“ schreibe, meine ich immer die linke Maustaste, sonst sag isch's euch.
64 Das Wort „desk“ kennt ihr aus dem Englischunterricht.
65 Wir haben bisher noch keine gelöscht, deshalb ist der Rubbish leer.
66 „Applications heißt auf deutsch „Anwendungen“.
67 Englisch „root“
68 Im englischen auch „slash“ genannt. Bei Windows wird übrigens der back-slash verwendet.
69 Englisch „folder“
70 Die Symbole die zu einem Verzeichnis oder einer Datei gehören werden auf englisch „icon“ genannt.

Leiste auf „View“. Jetzt öffnet sich ein Pulldown[71]-Menü in dem wir auf „Detailed List View[72]“ klicken. In dem großen Unterfenster des File-Managers wird jetzt jeder Unterordner und jede Datei in eine eigene Zeile einer Tabelle geschrieben. Klicken wir auf die erste Spaltenüberschrift „Name“, so wird die Reihenfolge der Zeilen umsortiert. Wir klicken hier noch mal und sehen jetzt die alphabetische Reihenfolge. Die Überschrift der zweiten Spalte heißt Description[73]. Hier steht zum Beispiel auch in der Zeile „home“ „folder“. Das überrascht uns jetzt nicht, denn wir wissen ja inzwischen, dass es sich bei „home“ um ein Verzeichnis handelt. Es gib noch mehr Spalten. Was die bedeuten findet ihr jetzt selbst heraus.

Klickt jetzt mal mit der rechten Maus-Taste auf das Verzeichnis „bin“. Es öffnet sich ein Auswahl-Menü in dem ihr ganz unten auf „Properties[74]“ klickt. Es öffnet sich ein neues Fenster und ihr werdet noch mehr Informationen zu dem Verzeichnis finden. Wenn ihr oben auf den stilisierten Karteikarten-Reiter mit der Aufschrift „Permission“ klickt, seht ihr ein neues Bild. Hier steht neben „Owner:[75]“ und „Group:[76]“ das Wort „root[77]“.

5.2 Die Rechte der LINUX- Benutzer

Und jetzt werden wir so langsam merken, dass LINUX eigentlich ein Betriebssystem für Großrechner ist. Ja, wir haben es auf dem kleinen Raspberry Pi mit einem Großrechner Betriebssystem zu tun, das zum Beispiel auch vom Deutschen Wetterdienst benutzt wird. Auf jedem LINUX-System gibt es wenigstens zwei Benutzer. Bei uns sind wir das mit dem Namen „pi“ und dann sind wir es noch mal mit dem Namen „root[78]“. Bei LINUX-Rechnern kann es beliebig viele Benutzer geben. Jeder Benutzer hat einen eigenen Bereich, in dem er seine Daten speichern kann. In seinem Bereich kann er festlegen, was man mit seinen Daten machen darf. Machen heißt bei LINUX „read“, „write“ oder „execute“[79]. Diese Rechte kann der „Owner[80]“ jeder Datei oder jedem Verzeichnis zuordnen. Der Benutzer „root“ hat eine Sonderstellung. Er kann mit allen Daten auf dem Rechner tun und lassen was er will, auch wenn er nicht der „Owner“ ist. Deshalb sollte auf einem LINUX-Großrechner das Passwort von „root“ auch nicht so einfach zu knacken sein.

Man kann mehrere Benutzer zu Gruppen zusammenfassen. Eine besondere Gruppe ist die Gruppe „root“ und ihr könnt euch jetzt sicher vorstellen, was diese Gruppe für Rechte hat. Und dann gibt es noch andere „Other“. Normalerweise vertraut man seinen Kollegen auf dem LINUX-Großrechner und gibt allen das Leserecht. Damit nicht jemand ungewollt die Daten verändern kann, gibt man nur sich selbst Schreibrechte. Deshalb sehen wir in den Ordnern, die unterhalb von unserem Verzeichnis „/home/pi“ sind, häufig unter Properties im File-Manager die „Access Control[81]“ Rechte „Read and

71 „pulldown“ heißt natürlich „herunter ziehen“. Herunterzieh-Menü ist aber ein furchtbarer Name.
72 Englisch „detaillierte Listen Ansicht“
73 Englisch für „Überschrift“
74 „Properties wird hier mit „Eigenschaften“ übersetzt.
75 Das ist der Besitzer des Verzeichnisses oder der Datei.
76 Das ist die Gruppe, zu der der Besitzer gehört.
77 Leider ist das Wort „root“ bei LINUX mehrfach besetzt. Erstens, und das haben wir schon kennen gelernt gibt es das Wurzelverzeichnis „root“. Dann gibt es aber auf jedem LINUX-(Groß und Klein)- Rechner einen Benutzer mit dem Namen „root“. Das ist der, der alles darf. Und dann gibt es noch eine Gruppe mit dem Namen „root“. Zu der gehört der Benutzer „root“.
78 Mit dem Befehl „sudo su“ im Terminal sind wir als User „root“ eingeloggt. Das erkennen wir jetzt daran, dass der Prompt jetzt „root@raspberrpi:/..#“ heißt. Probiert es aus, aber geht möglichst bald mit „exit“ wieder zurück, denn als „root“ kann man ziemlichen Unfug anrichten, wenn man nicht weiß was man tut, bis zum zerstören des LINUX Betriebssystems.
79 Lesen, schreiben und ausführen
80 Das ist der Besitzer der Datei.
81 Zugangsregelung

Write“ für den Benutzer „root“ und nur „Read only“ für alle anderen.

Das klingt jetzt etwas kompliziert, ist es auch, aber dadurch wird das System sehr sicher. Bei LINUX werdet ihr nicht so schnell Probleme mit Viren bekommen, denn der Virus wird nicht so schnell an die Rechte kommen, um Dateien des Betriebssystems zu verändern.

6 Mit Wi-Fi[82] ins Internet

Wir wollen jetzt mit dem Raspberry Pi ins Internet. Das geht mit dem Ethernet-Anschluss[83] und einem entsprechendem Kabel zu unserem Haus-Router[84], oder besser über WLAN[85]. Ich gehe mal davon aus, dass ihr einen WLAN-Server in eurem Haushalt habt. Falls nicht, solltet ihr jetzt einen zum Beispiel bei ebay ersteigern. Der Billigste tut es, denn egal wie schnell und teuer eure Verbindung von eurem Internet-Provider zu eurem Haus-Router ist, diese Verbindung ist eigentlich immer langsamer als die langsamste WLAN-Verbindung zwischen eurem Computer und dem Router und die langsamste Verbindung in der Kette bestimmt die Geschwindigkeit. Später werden wir noch einen LINUX-Labtop zum WLAN-Router machen, damit wir die Ameise auch ohne Haus-Router fernsteuern können. Aber auch da ist die Geschwindigkeit nicht wirklich wichtig. Ich lasse daher in der nachfolgenden Beschreibung alle Informationen weg die wir nicht brauchen.

Ich sollte noch erwähnen, dass der billige WLAN-Router als DHCP[86]-Server arbeiten sollte, damit sich Labtops und auch der Raspberry Pi automatisch anmelden können.

Als USB-WLAN-Adapter für den Raspberry Pi empfehle ich den EDIMAX oder den Wi-Pi[87]. Beide Adapter benutzen Treiber, die schon im LINUX-Kernel von raspbian enthalten sind und funktionieren plug-and-play[88]. Es gehen auch andere USB-WLAN-Adapter, aber nicht alle. Ihr solltet euch also besser vor dem Kauf informieren.

Nachdem wir den USB-WLAN-Adapter eingesteckt haben, starten wir auf dem Desktop des Raspberry Pi das Programm „wpa_gui“[89], indem wir auf das Symbol mit dem Namen „WiFi Config“ doppelklicken.

82 Wi-Fi ist ein Produktzertifikat für kompatible WLAN Verbindungen.
83 Der Ethernet-Anschluss ist links neben den USB-Anschlüssen.
84 Ich habe einen alte „Speedport W900V“ von der Telekom mit dem ich auch die DECT-Telefone betreibe.
85 WLAN ist die Abkürzung für „wireless local area network“, also ein „drahtloses Netzwerk im lokalen Bereich“.
86 DHCP steht für „dynamic host configuration protocol“.
87 Der EDIMAX ist sehr klein, dafür hat der Wi-Pi den besseren Empfang.
88 Einstecken und sofort damit arbeiten.
89 Wenn ihr auf „Help“ ganz oben klickt, erfahrt ihr, wem wir dieses tolle Progarmm verdanken.



Abbildung 11: Das Fenster des Programms "wpa-gui" mit dem Symbol "WiFi Config"

Ihr solltet jetzt das Fenster des Programms „wpa_gui“ sehen, wie in Abbildung 11. Das Programm kann auch von der Kommandozeile aus gestartet werden, indem man im Terminal-Fenster den Befehl „gui_wpa“ eingibt. Oben im Fenster steht rechts neben „Adapter:“ jetzt „wlan0“. Darunter steht neben „Network:“ der Name des WLAN-Netzwerks, mit dem wir uns verbinden wollen. Mein Netzwerk heißt „WLAN-huelsmann“, bei euch steht natürlich etwas anderes, oder erst mal gar nichts. Man kann sich mit mehreren Netzwerken verbinden, die dann durchnummeriert, beginnend mit 0, ausgewählt werden können. Daher steht hier noch „0:“ vor meinem Netzwerk-Namen. Darunter gibt es drei Karteikarten-Reiter mit den Namen „Current Status[90]“, „Manage Network“[91] und „WPS“. „Current Status“ sollte vorn liegen, sonst müsst ihr es anklicken.

Jetzt klickt ihr unten rechts auf den Button[92] mit dem Namen „Scan“ und dann seht ihr ein neues Fenster.

90 Momentaner Zustand
91 Netzwerk verwalten
92 Der deutsche Name für „Button“ ist „Schaltfläche“, aber den benutzt keiner.

Abbildung 12: Der Ergebnis des WLAN Scan

In Abbildung 12 erkennt ihr mein Ergebnis von Scan. Bei euch sieht das so ähnlich aus. Ich habe nur einen Eintrag von meinem WLAN-Netzwerk mit dem Namen „WLAN-huelsmann“[93]. Ihr findet hier jetzt eine Zeile mit dem Namen von eurem Netzwerk und möglicherweise auch noch die Namen der Netzwerke eurer Nachbarn. Ich habe diese Fenster etwas in die Breite gezogen, um alles zu sehen, denn hier stehen einige nützliche Informationen. Zum Beispiel arbeitet mein Netzwerk bei einer Frequenz von 2.457 GHz[94]. Die Signalstärke, mit der mein USB-WLAN-Adapter meinen WLAN-Router empfängt beträgt -73 dBm[95]. Das ist nicht viel, aber da könnten auch noch kleinere Werte stehen, bis der Empfang schlecht wird. In der Spalte „flags“ steht bei mir „[WPA-PSK-TKIP][WPA2-PSK-CCMP][ESS]“. Hier teilt uns mein WLAN-Router mit, wie die Übertragungsdaten verschlüsselt werden, damit kein anderer sie mitlesen kann. Ihr solltet diese Zeile doppelklicken damit ihr das folgende Fenster steht:

93 Falls hier gar nichts steht, solltet ihr es mit erneutem booten versuchen. Falls dann immer noch nichts da steht müsst ihr mal schauen, ob ihr den USB-wlan-Adapter über einen USB-Hub mit separater Stromversorgung anschließen könnt. Manchmal hilft auch den wlan-Adapter direkt in den Rapberry-Pi zu stecken und Maus und Tastatur über den Hub zu versorgen. Ich hatte jedenfalls mit manchen Pi-Platinen Schwierigkeiten und hab nicht herausbekommen warum. Irgendwann klappte es aber immer.

94 Gigahertz = GHz, also eine Milliarde Hertz. Die Einheit der Frequenz wird zu Ehren von Heinrich Hertz so genannt. Heinrich hat damals als Erster die von James Clark Maxwell vorhergesagten elektromagnetischen Wellen nachgewiesen. Ohne Heinrich hätten wir wohl etwas länger auf Radios, Fernseher und Handys verzichten müssen.

95 Das „dBm“ (deziBell em) ist eine sonderbare Einheit für eine Leistung, wird aber von Technik Nerds gern verwendet. Wegen Alexander Graham Bell, der die Verbreitung des Telefons voran gebracht hat, verwenden wir diese Einheit. So umstritten diese Einheit auch ist, Alexanders Ruf ist es übrigens auch, aber das könnt ihr selbst nachlesen. Jedenfalls hat das Telefon ihn reich gemacht. Übrigens, das kleine „m“ hinter dem „dB“ kennzeichnet, dass wir uns auf ein Milliwatt (1 mW = 0,001 Watt) beziehen. Streng genommen ist das dBmgar keine Einheit, sondern es kennzeichnet das logarithmisches Verhältnis der gemessenen Leistung zu einem Milliwatt. -73 dBm sind also -7,3 Bell bezogen auf ein Milliwatt oder $10^{-7,3}$ * 1 mW = 0,00000005 mW = 0,05 nW. Das ist eine ganz schön winzige Leistung und wir können daher Bauklötze staunen, wie hervorragend und trotzdem billig heute kleine Antennen-Verstärker sein können. Wenn man sich viel mit Verstärkern beschäftigt (wie ich), findet man plötzlich das dB wieder praktisch, weil man bei Verstärkerketten die dB's nur addieren muss, um die Gesamtverstärkung zu ermitteln.

Abbildung 13: Die Einstellungen für die Verbindung zum WLAN-Router

Wie ihr seht, werden die Einträge aus „Scan“ größtenteils in Abbildung 13 übernommen. Zum Beispiel steht bei „SSID:[96]“ der Name eures WLAN-Routers[97]. In der Zeile darunter steht hinter „Authentication[98]:“ eines der Verschlüsselungsverfahren, das mit „Scan“ gefunden wurde und in eckigen Klammern stand. Darunter steht hinter „Encryption:“ das Entschlüsselungsverfahren, das mit dem Scan[99] gefunden wurde und auch in der eckigen Klammer stand. Wir machen also nichts falsch, wenn wir hier nichts verändern. Jetzt müssen wir in „PSK:“ nur doch das Passwort unseres WLAN-Netzwerks eingeben. Wenn ihr das nicht kennt, müsst ihr denjenigen fragen, der den WLAN-Router eingerichtet hat. Dieses Passwort sollte nicht zu einfach und schon gar nicht leicht

96 SSID steht für „service set identification“. Das ist einfach der Name des WLAN-Netzwerks. Diesen Namen vergibt man, wenn man den WLAN-Router anlegt. Falls man bei der Einrichtung des WLAN-Routers keinen Namen angegeben hat, heißt er so, wir er vom Hersteller angegeben wurde. Eine SSID hat jedes WLAN-Netzwerk, sonst ist es keins.

97 Bei mir natürlich die SSID meines WLAN-Routers mit dem Namen „WLAN-huelsmann“.

98 Beglaubigung

99 „Scan“ heißt soviel wie absuchen.

erratbar sein[100]. Die anderen Einträge sind nicht weiter wichtig für uns, so dass wir das Fenster mit dem Klicken auf „Save[101]“ schließen können. Wir sehen jetzt wieder das Fenster von Abbildung 11. Hier drücken wir den Button „Connect“ und dann schauen wir, was sich tut. Der Raspberry Pi versucht sich jetzt mit dem WLAN zu verbinden. Das kann ein paar Augenblicke dauern. Dann sollte die Verbindung stehen. Bei „Status:“ sollte jetzt „Completed“ stehen. In „Authentication“, „Encryption:“, und „SSID“ stehen jetzt die Einträge, die wir schon kennen. In „BSSID:“ stehen jetzt Buchstaben und Zahlen, wobei immer zwei Ziffern durch einen Doppelpunkt getrennt sind. Das ist die MAC-Adresse[102] unseres USB-WLAN-Adapters. Als letzten Eintrag gibt es noch die „IP address:“. Hier steht 192.168.2.xxx, wobei xxx für eine Zahl steht, die der DHCP-Server des WLAN-Routers an euren Raspberry Pi vergeben hat.

Wir können jetzt das Programm „wpa_gui“ schießen. In der Taskleiste sehen wir jetzt ein Symbol des WLAN. Ein Klick auf das Symbol und wir sehen das Fenster des „wpa_gui“ wieder. Ab jetzt versucht der Raspberry Pi sich automatisch mit dem Internet zu verbinden, wenn wir ihn starten.

Das wir mit dem Internet verbunden sind, können wir jetzt mit dem Internet-Browser[103] Midori überprüfen. Das grüne Symbol des Midori liegt links auf dem Desktop und wir können den Browser durch einen Doppelklick starten.

Zusammenfassung:

1) Den USB WLAN-Adapter anschließen.
2) Den Raspberry Pi neustarten.
3) Das Programm Wi-Fi oder „wpa_gui“ starten.
4) Mit Scan den Router finden und die Einstellungen merken.
5) Doppelklick auf die Zeile in Scan, die unseren Router anzeigt.
6) Das Passwort eingeben.
7) Mit Connect den Raspberry Pi mit dem Router verbinden.
8) Mit dem Midori ins Internet.

7 Der Editor Emacs

Ein Editor ist ein Programm, das uns hilft andere Programme, aber auch andere Texte zu schreiben. Ich benutze immer gern den GNU[104] Emacs[105], der auf Richard Stallman[106] zurück zu führen ist.

100 Der Name des Haustiers als Passwort wäre zum Beispiel grob fahrlässig.
101 Haben wir das schon mal gemacht, steht hier jetzt add und wir legen statt „0: WLAN-Name“ jetzt „1: WLAN-Name“ an.
102 Die MAC-Adresse (Media Access Control) ist eine Kennung, die ein Netzwerk-Bauteil eindeutig identifiziert.
103 Wir kennen schon den Firefox oder den Internet-Explorer. Das sind auch beides Browser, die uns die Informationen von Webseiten anzeigen können.
104 GNU steht für das rekursive Akronym“GNU's Not Unix“. Unix ist ein Betriebssystem, dass als Vorbild für LINUX galt. Das ursprüngliche GNU Betriebssystem von Richard, siehe unten, war unbrauchbar, daher ist aus GNU heute GNU/LINUX oder einfach nur LINUX geworden.
105 Der Name des Editors Emacs steht für Editor MACroS. Macros sind eine Ansammlung von Befehlen an ein Programm, die nacheinander ausgeführt werden sollen.
106 Richard Stallman ist so ein richtiger Urcomputer-Hacker mit langen Haaren und etwas Bauchspeck. Er sitzt halt einfach zu viel vor dem Computer. Richard haben wir viel zu Verdanken, wenn es darum geht Software ohne Einschränkungen zu nutzen oder zu verändern. Das können wir nämlich nur, wenn wir dazu die Rechte haben und Richard kämpft schon immer für diese, unsere Rechte.

Emacs ist in unserem „raspbian“ Betriebssystem erst mal nicht enthalten. Wenn wir aber eine Verbindung zum Internet haben, können wir Emacs leicht nach installieren. Dazu starten wir ein LXTerminal und tippen auf der Kommandoebene den Befehl:

sudo apt-get install emacs

Um zu verstehen, was hier anläuft nehmen wir den Befehl mal auseinander. Mit dem vorangestellten Befehl „sudo“ verschaffen wir uns „root“-Rechte. Mit „apt-get“ verbinden wir uns mit der APT Bibliothek im Internet. Hier gibt es viele nützliche Programme, die schon so bereitgestellt sind, dass sie sich automatisch auf unserem Raspberry Pi installieren lassen. Mit dem Zusatz „install“ teilen wir der APT-Bibliothek mit, dass wir gerne ein Programm installieren würden. Jetzt müssen wir der Bibliothek nur noch mitteilen was wir haben wollen. Wir schreiben also dahinter „emacs“.

Schauen wir mal was passiert. Es gibt eine ganze Menge Informationen, die wir in Ruhe durchlesen. Unser Raspberry Pi unterhält sich nämlich mit der APT-Bibliothek darüber, welche Dateien uns fehlen, damit Emacs auch richtig installiert wird und fehlerfrei funktioniert. Irgendwann werden uns Fragen gestellt, die wir mit ja[107] beantworten sollten. Das ganze geht eine Weile. Wir können übrigens unten in der Task-Leiste rechts ein kleines Kästchen verfolgen, dass mit grünen Balken anzeigt, wie stark der Pi ausgelastet ist. Wenn die grünen Balken das Kästchen von unten bis oben ausgefüllt haben, ist der arme Pi voll am Rödeln und wir sollten einfach abwarten. Die Mitteilungen im LXTerminal können das ganze Fenster füllen. Irgendwann sehen wir aber wieder den Prompt[108]. Unser Prompt sollte übrigens etwa so aussehen:

pi@raspberrypi ~$

Jetzt geben wir hinter dem Prompt „emacs“ ein und sollten das folgende Fenster sehen:

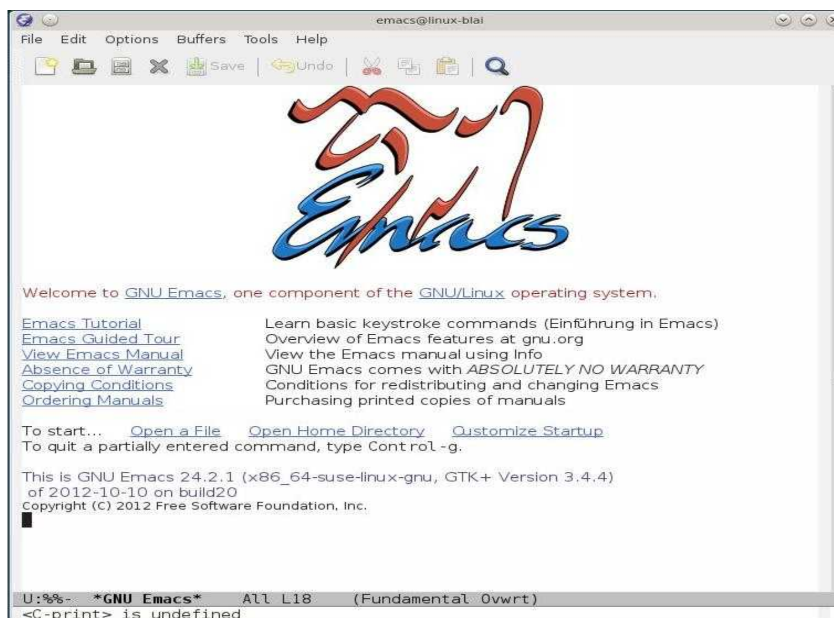


Abbildung 14: Das Startfenster des GNU Emacs

Nur um etwas mit dem Emacs vertraut zu werden schreiben wir einen blöden Text in eine Datei. Dazu gehen wir mit dem Mauszeiger oben links im Emacs Fenster auf das stilisierte Blatt-Papier-

107 Y oder y für yes.

108 Der Prompt steht am Anfang einer Kommandozeile, wenn wir neue Befehle eingeben können.

Symbol mit dem kleinen grünem Plus und schauen uns das Popup[109] an. In dem Popup steht „Specify a new file's name, to edit the file[110]“. Wenn wir nicht schnell genug lesen, und das kann passieren, verschwindet die Meldung wieder. Wir klicken also auf das Papier-Symbol und es öffnet sich ein neues Fenster, das wir so ähnlich schon mal beim „File Manager“ gesehen habt. Oben in der Eingabezeile neben „Name:“ blinkt der „Cursor[111]“. Hier geben wir den Namen unserer ersten Textdatei an. Ich habe hier phantasieloser Weise den File-Namen „text1“ eingegeben und unten rechts auf den Button geklickt, der mit einem grünen Kreis in dem ein weißer Pfeil versehen ist und auf dem außerdem noch „OK“ steht.

Nun verschwindet das Fenster und es öffnet sich ein neues Fenster mit einer großen weißen Fläche, in die wir etwas schreiben können. Ich habe hier so phantasielos wie oben erwähnt „Dies ist mein erster Text“ eingegeben. Danach habe ich auf das Symbol der Diskette geklickt, um die Datei „text1“ zu speichern. Das Symbol ist links neben dem orangen Symbol mit dem „x“ und wenn ihr mit dem Mauszeiger darüber fahrt, kommt wieder so ein Popup und erklärt euch was passiert. Interessanter Weise wird das Disketten-Symbol jetzt grau, aber sobald wir wieder etwas an unserem Text verändern wird es wieder normal. Der Emacs ist ganz schön gut programmiert, findet ihr nicht auch?

Wir schließen den Emacs jetzt wieder und öffnen den File-Manager unten in der Task-Leiste. Jetzt sollten wir in unserem Hauptverzeichnis /home/pi nicht nur die Verzeichnisse „python_games“ und „Desktop“ finden, sondern auch die neu erzeugte Datei „text1“. Wenn ihr hier jetzt einen Doppelklick auf „text1“ macht, öffnet sich der Emacs und zeigt uns den Inhalt von „text1“ an, damit wir an dem Text weiter arbeiten können. Aber nach dem Doppelklick nur Geduld, der Pi ist ja kein Supercomputer; obwohl er mit LINUX läuft.

8 Kleine Einführung in die Computersprache c/c++

Eigentlich wurde von den Machern des Raspberry Pi die Computersprache Python für erste Programmierkontakte vorgesehen. Python Programme laufen aber einfach zu langsam. Mich stört so was. Ich bin daher der Meinung, wir sollten uns mit Python nicht aufhalten und gleich mit der Computersprache beschäftigen, mit der übrigens auch Betriebssysteme programmiert werden. C ist eine Hardware nahe Computersprache, das heißt, das man damit sehr effiziente Treiber für Hardware schreiben kann. Unser Ziel ist Hardware zu programmieren. Wir wollen Servos ansteuern und unserer Roboter-Ameise das Laufen beibringen. Auf dem Weg dorthin lernen wir ganz nebenbei C. Die Spracherweiterung von C hat übrigens von Bjarne Stroustrup[112] den Namen C++ bekommen, aber wir werden diese Erweiterung nur da benutzen, wo wir sie brauchen. Wer mehr über C/C++ wissen will, und das kann passieren, wenn man erst mal von der Programmierung infiziert wurde, sollte sich ein gutes Buch[113] besorgen.

Fangen wir mit dem dümmsten Programm an, mit dem wohl jeder angehende Programmierer, egal welche Programmiersprache er lernt, in Berührung kommt. Diese Programm heißt „HelloWord“. Das Programm besteht aus Nullen und Einsen und nur der Raspian Pi wird es lesen können, wir

109 Unter einem Popup versteht man eine Meldung in einem kleinen Kästchen mit einer kleinen Information.
110 Bestimme einen neuen Datei-Namen, um die Datei zu ergänzen.
111 Unter „Curser“ (ausgesprochen „Körser“) versteht man hier einen kleinen senkrechten blinkenden Strichs, an dessen Stelle man Zeichen eingeben kann.
112 Bjarne war Programmierer bei den Bell Labs als er c zu c++ erweiterte. Heute ist er Professor in Texas. Bjarne ist nicht nur der Vater von c++, sondern bringt uns sein Baby auch näher. Es gibt einige ganz brauchbare Bücher von ihm, teilweise auch in deutscher Übersetzung.
113 Ganz gut finde ich zur Einführung Thomas Wielands Buch „C++ Entwicklung mit LINUX“. Vielleicht sollte Thomas mal wieder an einer neuen Auflage arbeiten.

nicht! Wir können es eigentlich noch nicht mal schreiben. Wir können nur einen Sourcecode[114] schreiben, den kann aber der Pi wiederum nicht verstehen. Deshalb benötigen wir einen Übersetzer, der wird Compiler genannt. Der Compiler ist wiederum ein Programm. Nur nicht verwirren lassen!

Wir benutzen den gcc[115]. Der gcc ist eigentlich bei allen LINUX-Distributionen dabei. Falls er bei unserer Distribution weggelassen[116] wurde, holen wir ihn uns aus dem Netz mit dem Befehl, den wir ja schon kennen und im LXTerminal als Konsolen-Kommando eingeben:

sudo apt-get install gcc

Um etwas Ordnung zu halten werden wir ein neues Verzeichnis anlegen, in das wir alle Dateien unseres Programms „HelloWord“ hinein legen. Ein neues Verzeichnis können wir zum Beispiel mit dem File-Manager aus der Task-Leiste anlegen. Dazu klicken wir, nachdem wir den File-Manager geöffnet haben, mit der rechten Maustaste in die große Fläche, in der unsere Ordner und Dateien liegen. Es kommt ein Popup in dem wir auf „Create New … „ drücken und dann „Folder“ auswählen. Wir werden nun aufgefordert mit dem Satz: „Enter a name for the newly created folder[117]:“ Das machen wir. Ich habe mein neues Verzeichnis „c++Progs“ genannt. Dieses erscheint jetzt im File-Manager und wir öffnen es mit einem Doppelklick. Das Verzeichnis „c++Progs ist erwartungsgemäß erst mal leer. Mit Rechtsklick in die Leere und der Prozedur mit der wir das Verzeichnis erzeugt haben, erzeugen wir jetzt einen neuen „Blank File[118]“ mit dem Namen „Hello_world.cpp“. Wenn wir jetzt mit einem Rechtsklick auf das Symbol von „Hello_world.cpp“ klicken, öffnet sich ein Popup und wir klicken auf „GNU Emacs 23“ wie in Abbildung 15 zu sehen ist.

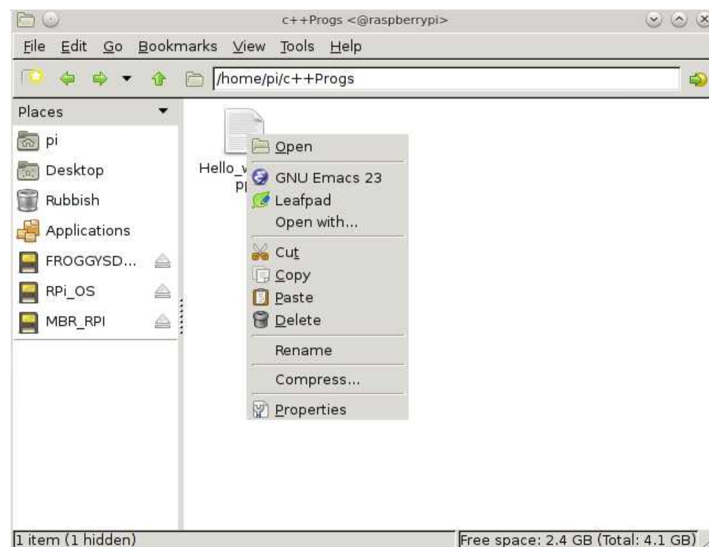


Abbildung 15: File Manager mit Auswahlmenü

114 „Source“ heißt „Quelle“ und als „code“ bezeichnet man den Text der Programmiersprachen-Kommandos.
115 Gcc steht für „GNU compiler colection“, also soviel wie „LINUX Programmiersprachen Übersetzer Sammlung“. Der gcc kann nämlich nicht nur c/c++ code übersetzen, sondern auch viele andere Programmiersprachen.
116 Das kann aus Platzgründen durchaus sein.
117 Gib einen Namen für das neu angelegte Verzeichnis ein!
118 „Blank File“ ist eine Datei in der nichts, aber auch gar nichts drin steht.

8.1 Den Sourcecode eines einfachen Programms schreiben und verstehen

Nach etwas Geduld öffnet sich der Emacs und wir können damit beginnen unser erstes, und zugegeben, etwas dummes Programm zu schreiben. Die Nummerierung am Zeilenanfang tippen wir nicht mit ein. Wir schreiben folgenden Sourcecode:

```
1  // Dateiname:       Hello_world.cpp
2  // Programmierer:   Axel
3  // Datum:
4  // Version:         0.0
5
6  # include <stdio.h>
7
8  main()
9  {
10    int A;
11    A = 815;
12    printf(“Hallo Welt\n“);
13    printf(„%i“,A);
14 }
```

Die Zeilennummerierung am Anfang sind nur Bezugsnummern und werden im Programmcode nicht eingegeben. Gehen wir den Sourcecode mal Zeile für Zeile durch.

Die Zeile 1 beginnt mit zwei Schrägstrichen. Alles was in einer Zeile hinter zwei Schrägstrichen steht hat für den Computer keine Bedeutung. Es steht da nur für uns. Bis zur Zeile 4 stehen da also nur Informationen die dazu dienen, dass wir auch nach Jahren diesen Sourcecode zuordnen können. Die Zeile 5 ist eine Leerzeile und hat auch keine Bedeutung und dient nur der besseren Lesbarkeit.

Die Zeile 6 beginnt mit einem “#“[119]. Dahinter steht mit „include[120]“ eine Anweisung an den Compiler. Dieser soll den Programmteil „stdio.h“ aus einer Programmbibliothek zu benutzen. Er steht in spitzen Klammern. Das bedeutet, das der Compiler selbst weiß, wo er ihn findet. Der Programmteil „stdio.h“ regelt die Standard Ein- und Ausgabe[121] und macht unser Programm unter anderem mit der Funktion „printf“ bekannt, aber dazu später mehr. In unserem Fall ist das die Ausgabe auf den Bildschirm.

In Zeile 8 beginnt das Hauptprogramm mit dem Namen „main“. Jedes c/c++ Programm hat ein Hauptprogramm, von dem alles ausgeht, sonst ist es kein c/c++ Programm. Direkt hinter „main“ ist eine offene und eine geschlossene runde Klammer. Zwischen diesen Klammern kann man Parameter übergeben. Wir nutzen diese Möglichkeit hier nicht. Nach dem Aufruf des Hauptprogramms „main()“ folgt eine geschweifte offene Klammer, hier in Zeile 9.

119 „#“ wird als Doppelkreuz oder Gartenhark bezeichnet.
120 „include“ heißt „einfügen“
121 „stdio“ steht für „standard input output“

Die Zeile 10 und folgende sind mit einem „;“ abgeschlossen. Damit wird angezeigt, dass der Befehl abgeschlossen ist. Der häufigste Fehler von c/c++ Anfängern ist das falsche Anwenden des “;“[122]. In c/c++ ist ein Befehl nämlich keinesfalls durch eine neue Zeile abgeschlossen. Ihr könntet auch mehrere Befehle in eine Zeile schreiben und jeweils durch ein „;“ trennen. Ihr könnt aber auch einen Befehl auf mehrere Zeilen verteilen.

Die Zeile 10 ist unter Verwendung der Tabulatortaste etwas eingerückt. Das ist guter Schreibstiel, , den wir uns angewöhnen, aber unnötig. Damit drückt man aus, dass alle Befehlszeilen mit der gleichen Einrückung zusammen gehören. Jetzt folgt die erste richtige Befehlszeile des Programms.

Wir definieren eine Variable mit dem Namen „A“. Vor „A“ steht das Wort „int“. Damit zeigen wir an, das die nachfolgende Variable vom Type Integer[123] ist. Jede Variable[124] muss einen Type haben, denn damit ist geregelt, wie viel Speicherplatz die Variable bekommt (ah, ja, die Hardware[125]-Nähe von C …) . In Zeile 11 wird der Variablen „A“ der Wert „815“ zugewiesen.

In Zeile 12 steht der Befehl „printf[126]“, der dafür sorgt, das unser Text in Hochkomma auf den Bildschirm ausgegeben wird. Der Backslash gefolgt von dem „n“ gibt an, dass Nachfolgendes in einer neuen Zeile ausgegeben werden soll.

In Zeile 13 geben wir den Inhalt der Variablen „A“aus. Dazu müssen wir der Funktion „printf“ mitteilen, wie wir es gerne hätten. Da wir eine Integer-Variable ausgeben wollen, verwenden wir die Formatanweisung „%i“. Formatanweisungen stehen immer in Hochkomma.

Jedes Programm und damit auch unser Hauptprogramm „main“ wird eigentlich mit dem Befehl „return[127]“ beendet. Wir lassen ihn weg, denn der gcc ist manchmal genauso nachlässig wie wir. Danach folgt die geschlossene geschweifte Klammer und unser Programmcode ist fertig.

Wir speichern das ganze im Emacs, verlassen den Emacs und kehren wieder auf die Komandozeile im LXTerminal zurück, um ein paar hilfreiche LINUX-Befehle zu lernen.

8.2 Mit der bash vertraut werden

LINUX-Befehle werden von einem Programm verarbeitet das „bash“ heißt. Die „bash[128] “ startet automatisch sobald wir das LXTerminal geöffnet haben und wartet auf unsere Befehle. Wir geben einfach mal den Befehl „hallo“ ein. Jetzt meldet sich die bash mit: „-bash: hallo: command not found“. Ja hallo, „hallo“ ist kein LINUX Kommando, das die bash versteht.

Wir geben daher erst mal „ls[129]“ ein mit der Option „-l[130]“ und siehe da, die bash versteht uns und

122 In c/c++ wird ein Befehl nicht durch eine neue Zeile abgeschlossen. Ihr könntet auch mehrere Befehle in eine Zeile schreiben und jeweils durch ein „;“ trennen. Ihr könnt aber auch einen Befehl auf mehrere Zeilen verteilen.

123 Integer Typen sind ganze Zahlen.

124 Eine Variable ist eine Speicherstelle in der ein Wert abgelegt ist, der auch geändert werden kann.

125 In der Computerei wird alles, was man irgendwie anfassen kann mit Hardware bezeichnet. Umgekehrt ist Software der Überbegriff für Programme und Daten.

126 „print steht für „drucken und mit „printf“ drucken wir eine formatierte Ausgabe auf die Konsole.

127 „return“ heißt hier soviel wie: Geh zurück wo du hergekommen bist.

128 „bash“ ist die Abkürzung für „bourne again shell“, also die wiedergeborne Shell. Nein, das hat hier nichts mit Matt Damon zu tun. Unter einer Shell versteht man ein Programm, mit dem man Befehle an das Betriebssystem schickt. Bei LINUX ist bash absoluter Standard. Es gibt aber auch andere Shells.

129 Der Befehl „ls“ steht für „list“ und listet alles auf, was im aktuellen Verzeichnis steht.

130 Befehlsoptionen werden bei LINUX-Kommandos häufig mit einem Minuszeichen und einem Buchstaben an den Befehl angehängt. „-l“ sagt hier, dass wir mehr Infos haben wollen. Probiert es mal ohne „-l“ aus und ihr wisst was ich meine. Wenn ihr wissen wollt, welche Optionen ein Befehl hat (kein Mensch kann sich die alle merken), dann gebt mal zum Beispiel „man ls“ ein und schaut was passiert. „man“ steht hier übrigens für „Manual“ oder Anleitung.

gibt einige Meldungen im TXTerminal aus.

```
pi@raspberrypi ~ $ ls -l
total 20
drwxr-xr-x 2 pi pi 4096 Jan 23 21:18 c++Progs
drwxr-xr-x 2 pi pi 4096 Jan  5 21:20 Desktop
drwxrwxr-x 2 pi pi 4096 Jul 20  2012 python_games
-rw-r--r-- 1 pi pi   26 Jan 22 21:19 text1
```

Wir sehen jetzt was im Verzeichnis /home/pi steht. Doch woher wissen wir, das es das Verzeichnis /home/pi ist? Ganz einfach am Prompt. Der Prompt sagt uns das wir der Benutzer pi auf dem Computer raspberrypi sind. Da hinter kommt ein Wellenzeichen. Dies deutet an, dass wir uns im Hauptverzeichnis des Benutzers pi befinden, und das ist nun mal /home/pi.

Die erste Zeile mit „total 20“ soll uns nicht weiter stören. Der erste Buchstabe in der zweiten Zeile ist ein „d“ und bedeutet, dass es sich bei c++Progs um ein „directory[131]“ handelt. Jetzt kommen einige „rwx“ oder „-“. Diese stehen für „read, write und execute“, also lesen schreiben und ausführen. Die Ersten drei stehen für die Berechtigung des „owner[132]“. Das sind wir. Die nächsten drei stehen für die „group[133]“ und die letzten drei für alle. Die Zahl, die dann folgt ist erst mal unwichtig. Dann kommt der Name des Besitzers („owner“), das sind wir und der Name der Gruppe, die heißt hier wie der Besitzer[134]. Dann kommt eine Zahl die die Größe in Byte[135] angibt. Danach kommt das Datum mitsamt der Uhrzeit, die angeben, wann das Verzeichnis oder die Datei das letzte mal verändert wurde Und zum Schluss steht da noch der Name der Datei oder des Verzeichnisses. Diese Informationen konnten wir uns auch vom „File Manager“ besorgen, da mussten wir dann ein bisschen rumklicken. Ihr seht, wenn man sich mit den LINUX-Befehlen auskennt, ist man auf der Kommandozeile schneller. Kennt man sich weniger aus, ist man mit klicken schneller.

Wir wechseln jetzt auf der Kommandoebene in das neue Verzeichnis „c++Progs“. Dazu benutzen wir den Befehl „cd c++Progs“[136]. Der Prompt hat sich jetzt verändert und zeigt uns an, das wir tatsächlich gewechselt haben. Mit „ls“ oder „ls -l“ können wir uns jetzt ansehen, was in dem Verzeichnis ist. Übrigens mit dem Befehl „cd ..“ kommen wir wieder in das übergeordnete Verzeichnis zurück.

8.3 Ein c/c++ Programm compilieren und ausführen

Wir haben also jetzt einen Sourcecode mit dem Emacs geschrieben und in dem Verzeichnis „c++Progs“ unter dem Namen „Hello_world.cpp“ abgespeichert. Wir wechseln in das Verzeichnis „/home/pi/c++Progs“ und geben hier den Befehl gcc Hello_world.cpp ein. Wenn alles richtig ist, sollten wir nach kurzer Zeit wieder den Prompt erhalten. Falls wir irgend etwas falsch gemacht haben, kommen jetzt Fehlermeldungen, die wir uns durchlesen sollten, da das hilfreich sein kann,

131 „directory“ ist hier das gleiche wie „folder“ und daher wissen wir, dass es ein „Verzeichnis“ oder „Ordner“ ist.
132 Das ist der Besitzer der Datei.
133 Da LINUX ein Großcomputer-Betriebssystem ist, könnten auch mehrere Benutzer mit dem Computer arbeiten. LINUX lässt es zu, diese zu Gruppen zusammen zu fassen. Auf dem Pi macht das aber wenig Sinn.
134 Nur so zur Verwirrung
135 Ein Byte besteht aus 8 bit. Ein bit ist die kleinste Speichereinheit und kann entweder 0 oder 1 sein. Alles andere versteht ein Computer nicht.
136 „cd“ steht für „change directory“, also „wechsle das Verzeichnis“.

um den Fehler zu finden, auch wenn sie auf Englisch sind. Wir können jetzt mit dem Emacs den Programmcode noch mal aufrufen, verändern und wieder abspeichern, ohne den Emacs zu schleißen. Dann wiederholen wir auf der Kommandoebene den Befehl „gcc Hello_world.cpp“. Das machen wir so lange bis wir alle Fehler beseitigt haben.

Jetzt geben wir auf der Kommandoebene ls -l ein, und wir sollte eine neue Datei sehen, die den Namen „a.out“ hat. Wenn wir dem gcc nämlich nicht mitgeteilten, wie das Programm heißen soll, das aus unsrem Sourcecode compiliert werden soll, gibt er ihm einfach den Namen „a.out“. Das es sich bei a.out um ein ausführbares Programm handelt, erkennen wir daran, dass es in der Liste der Einträge, die durch „ls -l“ ausgegeben wird, grün erscheint. Außerdem erkennen wir die „x“ Einträge am Anfang der Zeile.

Um das Programm auszuführen geben wir einfach „./a.out“ ein und schauen was passiert. Da sollte jetzt in der nächsten Zeile „Hallo Welt“ stehen und dann „pi@raspberrypi ~/c++Progs $“. Warum, das solltet ihr jetzt verstanden haben, oder ich habe es schlecht erklärt.

Um das Programm a.out umzubenennen können wir wieder wieder im File-Manager ein bisschen rumklicken. Oder wir benutzen den LINUX-Befehl „mv“[137]:

mv a.out HelloWorld

Mit „rm“[137] könnt ihr Dateien auch wieder löschen. Damit wir aber den Umweg über das umbenennen gar nicht erst gehen müssen, können wir dem gcc auch mit der Option „-o“ direkt sagen, wie unser Programm heißen soll:

gcc -o HelloWorld Hello_world.cpp[138]

Zusammenfassung:

1) Mit dem Emacs den Sourcecode schreiben und unter ~/c++Progs/Hello_world.cpp abspeichern.
2) Im LXTerminal mit cd c++Progs ins Verzeichnis /home/pi/c++Progs wechseln.
3) Mit ls -l den Sourcecode Hello_world.cpp finden.
4) Mit gcc -o HelloWorld Hello_word.cpp den Sourcecode compilieren.
5) Fehler beseitigen mit dem Emacs.
6) Mit ls -l das Programm HelloWorld finden.
7) Mit ./HelloWorld das Programm ausführen.

8.4 Makefiles

Makefiles werden erst so richtig praktisch, wenn wir Programme schreiben, die aus mehreren Sourcecode-Dateien bestehen[139]. Programm-Datei, Sourcecode, Beschreibungstextfile und Makefile packt man vorteilhafter Weise in einen eigenen Projekt-Ordner. Wir erzeugen einen in

137 „mv“ steht für „move“, „bewegen“. Versucht mal: „mv a.out /home/pi/a.out“ und schaut wo die Datei geblieben ist, dann wisst ihr, was ich mit bewegen meine. Ein weiterer LINUX-Befehl ist „rm a.out“. „rm“ steht für „remove“ Damit seid ihr die Datei wieder los. Ihr könnt die Datei ja jederzeit mit dem gcc wieder neu erzeugen.

138 Ich habe hier natürlich absichtlich ähnliche Namen genommen, damit man weiß, dass die Dateien zusammen gehören. LINUX unterscheidet zwischen Groß- und Kleinschreibung. Manche Fehlermeldung ist auf nachlässige Schreibweise zurück zu führen, aber ihr seid ja sorgfältig, oder?

139 Für so kleine „HelloWorld“ Programme sind Makefiles völlig überflüssig, wie wir gesehen haben, aber wir wollen ja später noch nettere Programme schreiben, und da ist es gut, wenn wir die Grundzüge schon mal kennen.

unserem Unterverzeichnis „c++Progs“ unseres Heimatverzeichnises „/home/pi/“ und geben ihm den Namen „ErstesProjekt“. Dazu können wir wieder in dem File-Manager rumklicken, oder auf der Kommandoebene den Befehl:

mkdir /home/pi/ErstesProjekt[140]

eingeben. In diesen Ordner kopieren wir jetzt die Datei „Hello_world.cpp“ mit dem LINUX-Befehl:

cp /home/pi/c++Progs/Hello_world.cpp ~/c++Progs/ErstesProjekt/[141]

Der Grundaufbau des Befehls „cp“ ist immer: „cp zuKopierendeDatei.Zusatz[142] woSollSieHin“. Dazwischen ist ein Leerzeichen. Bei der „zuKopierendenDatei“ habe ich den kompletten Pfad mit angegeben. Bei der „woSollSieHin“ Datei habe ich mir Erleichterung verschafft, in dem ich mein Homeverzeichnis mit „~“ angeben habe und den Dateinamen weggelassen habe. Das Programm „cp“[143] weiß ja schon wie die Datei heißt und bietet mir den bescheidenen Komfort an, den Dateinamen wegzulassen.

Aber zurück zum Makefile. Diesen erstellen wir uns natürlich auch mit dem Emacs und speichern ihn in unserem Projekt-Ordner[144] „ErstesProjekt“. Es ist altes Brauchtum dem Makefiles immer den Namen „Makefile“ zu geben, dann muss keiner lange danach suchen. In dem Makefile stehen die Abhängigkeiten und Regeln, die angeben, wie die Programm-Dateien erzeugt werden sollen, mit allen Optionen an den Compiler und Linker[145]. In jeder Zeile steht eine Anweisungen. Abhängigkeiten stehen ganz links am Anfang der Zeile, auszuführende Regeln müssen mit dem Tabulator[146] eingerückt werden. Emacs erkennt die Syntax[147] eines Makefiles und hilft uns durch farbiges Hervorheben. Kommentare beginnen immer mit einem vorangestellten Doppelkreuz #. Für unser HelloWorld Programm sieht der Makefile so aus:

```
# Dies ist der Makefile für das Programm HelloWorld
# Erstellt von Forrest Black am 31.02.2013
# Version 0.1
HelloWorld: Hello_world.cpp
	gcc -o HelloWorld Hello_world.cpp
```

Wir verlassen jetzt den Emacs nachdem wir den Makefile gespeichert haben und gehen im TXTerminal auf die Kommandozeile zurück. Wenn wir in unserem Projekt-Ordner sind, sehen wir das an unsrem Prompt. Der sollte jetzt so aussehen:

pi@raspberrypi ~/c++Progs/ErstesProjekt $

140 „mkdir“ steht für „make directory“, also: Mach ein Verzeichnis.
141 „cp“ steht für „copy“. Soviel englisch kann jeder, oder?
142 Der „.Zusatz“ wird auf englisch „extension“ genannt und wir erkennen an ihm mit welchem Programm wir die Datei bearbeiten können.
143 bash Befehle sind meistens selbst Programme.
144 Unter Ordner, Verzeichnis oder auch englisch „folder“ versteht man immer das gleiche. Lasst euch dadurch nicht verwirren.
145 Der Linker ist ein Programm, dar mehrere Dateien, die vom Compiler übersetzt wurden, zusammen bindet, falls das notwendig ist..
146 Leerzeichen tun hier nicht das gleiche und sind oft die Quelle von Fehlern.
147 Die „Syntax“ ist sozusagen die Grammatik einer Programmiersprache und auch der Makefile wird in der eigenen Makefile Sprache geschrieben, die von dem Programm „make“ verstanden wird.

Mit ls -l können wir wieder nachschauen, ob unser Sourcecode „Hello_world.cpp“ und unser Makefile auch wirklich da sind. Falls ihr Sie nicht findet, müsst ihr euch auf die Suche begeben, oder nochmal probieren sie anzulegen.

Jetzt ist es an der Zeit euch das Programm „make“ vorzustellen. Das Programm „make“ startet den gcc und befolgt dabei die Anweisungen, die im Makefile stehen. Das ist schon alles. Ihr gebt also einfach hinter dem Prompt „make“ ein. Und schon wird euer Sourcecode Hello_world.cpp übersetzt und die Programm-Datei HelloWorld erzeugt. Wen überrascht das noch? Mit ls -l sehen wir trotzdem nach. Unser gerade erzeugtes Programm können wir jetzt wieder mit ./HelloWorld[148] ausführen.

Gar nicht schlecht, oder? Aber es kommt noch besser. Wir können nämlich, wenn wir ein Makefile haben, das Compilieren auch mit dem Emacs starten.

Dazu öffnen wir den File-Manager[149] und klicken uns zum Verzeichnis ../ErstesProjekt durch. Hier sehen wir jetzt drei Dateien: HelloWorld, Hello_world.cpp und Makefile. Vielleicht sehen die Symbole bei euch leicht anders aus, als in Bild 16. Das macht nichts.

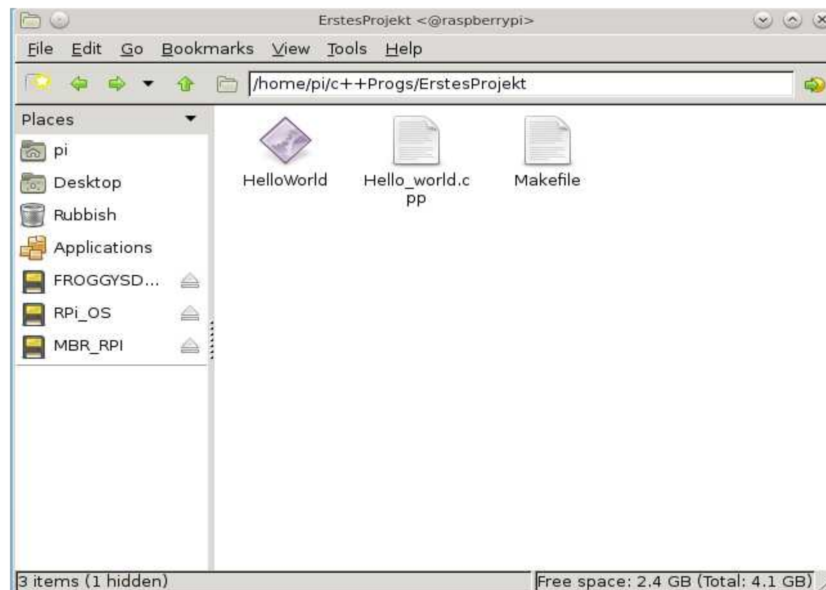


Abbildung 16: Der File-Manager mit dem Inhalt des Ordners: ErstesProjekt

Starten wir also wieder mit Rechtsklick auf die Datei „Hello_world.cpp“ den Emacs und editieren sie, bis sie aussieht wie in Bild 17. Worauf es ankommt, ist eigentlich nur die letzte Zeile, in die ich jetzt im printf bei der Format-Anweisung für die Integer-Zahl ein „\n“ eingefügt habe. Dadurch sollte jetzt nach der Ausführung des Programms „HelloWorld“ der Prompt in einer neuen Zeile stehen. Wir klicken erst mal oben in der Leiste auf „Tools“. Es öffnet sich ein Pulldown-Menü in dem wir auf „Compile ..) klicken. Jetzt erscheint ganz unten im Emacs eine Zeile in der steht:

148 Übrigens geben wir mit dem „./“ vor dem Prorammnamen nur an, das wir es aus dem Verzeichnis starten wollen, in dem wir gerade sind.

149 Den File-Manager können wir auch bequem von der Kommandozeile aus öffnen mit dem Befehl: pcmanfm. Wenn ihr im File-Manager auf Help klickt, wisst ihr, woher ich diesen Befehl habe.

Compile command: make -k

Hier drücken wir einfach die „weiter“ Taste[150]. Jetzt werden wir gefragt:

Save file /home/pi/c++Progs/ErstesProjekt/Hello_world.cpp (y, n, ! , …)

Wir geben natürlich „y“ für yes ein. Der Compiler macht jetzt genau das was im Makefile steht. Das teilt Emacs uns auch in dem mittleren Fenster mit. Wenn wir das ganze wiederholen, stellt das Programm „make“ fest, das nichts zu tun ist mit dem Satz: 'HelloWorld' is up to date.

```
emacs23@raspberrypi <@raspberrypi>
File Edit Options Buffers Tools C++ Help
/*
Pogramm Name: hello_pi.cpp
Progarmmierer: pi
Version: 2013-01-20

Dies ist ein ganz simples Programm mit dem wir feststellen,
ob der gcc richtig arbeitet.
*/

#include <stdio.h>

main()
{
  int A;
  A = 815;
  printf("Hello\n");
  printf("%i\n",A);
}
--:**-  Hello_world.cpp   All L17    (C++/l Abbrev)----------------------
-*- mode: compilation; default-directory: "/home/pi/c++Progs/ErstesProjekt/" -*-
Compilation started at Sat Jan 26 16:27:44

make -k
make: `HelloWorld' is up to date.

Compilation finished at Sat Jan 26 16:27:44
-U:%*-  *compilation*   All L3     (Compilation:exit [0])------------------
```

Abbildung 17: Die veränderte Datei "Hello_world.cpp" im Emacs, nachdem sie compiliert wurde.

Eigentlich wollte ich euch in diesem Kapitel nur etwas c/c++ beibringen. Jetzt habt ihr auch noch etwas bash, LINUX und make gelernt. Außerdem könnt ihr jetzt den Emacs als Programmierhilfe[151] nutzen. Nicht schlecht für den Anfang!

9 Der GPIO Stecker des Raspberry Pi

Auf der Platine des Raspberry Pi befindet sich eine Stiftleiste mit 26 Stiften. Die Stiftleiste hat die Bezeichnung „P1“ und befindet sich neben der gelben Video-Buchse. Bild 18 zeigt das entsprechende Detail der Platine.

150 Die „weiter“ Taste wird auch häufig mit „Return“ bezeichnet oder auch „neue Zeile“.
151 Ein Programm, das uns mit einer grafischen Benutzeroberfläche beim Programmieren hilft, wird auch IDE genannt. IDE steht für „integrated development environment“ oder „integrierte Entwicklungsumgebung“.

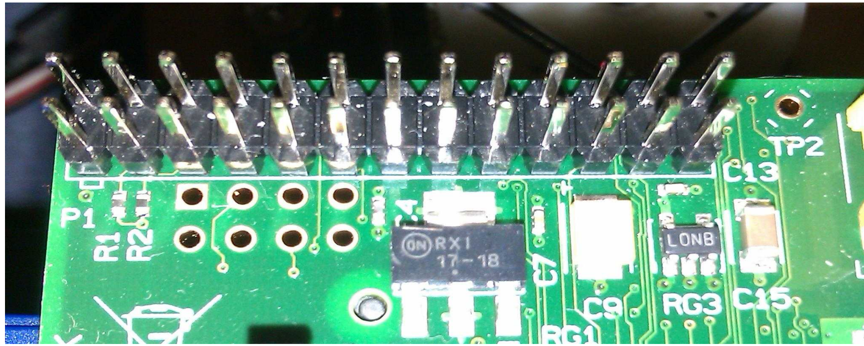


Abbildung 18: Die 26-polige Stiftleiste mit der Bezeichnung P1. Hier finden wir die GPIOs.

Auf der Stiftleiste befinden sich einige GND[152] pins, +5 Volt und +3,3 Volt Spannungen und die GPIO[153]s. Es gibt inzwischen mehrere Versionen des Raspberry Pi. Als ich dieses Buch schrieb, war die Revision 2 die neueste. Leider hat sich dadurch die Pin-Belegung leicht geändert. Die Bedeutung der einzelnen Pins ist in Tabelle 1 definiert. Pin 1 ist auf der Platine dadurch besonders gekennzeichnet. Das Löt-Pad ist nicht rund, sondern quadratisch ist.

152 GND steht für „ground“. Auf deutsch wird das mit „Masse“ oder „Erde“ übersetzt. Damit ist das Bezugssystem der Spannungen gemeint. GND ist das Bezugs-Potential und hat die Spannung 0V.

153 GPIO steht für „general surpose input output“. Gemeint sind damit frei programmierbare Anschlüsse zur Ein- und Ausgabe.

Pull	Rev. 1	Rev. 2	Pin			Pin	Rev. 2	Rev. 1	Pull
	+3,3 V	+3,3 V	1	●	●	2	+5 V	+5 V	
high	GPIO0	GPIO2	3	●	●	4	+5 V	DNC	
high	GPIO1	GPIO3	5	●	●	6	GND	GND	
high	GPIO4	GPIO4	7	●	●	8	GPIO14	GPIO14	low
	DNC	GND	9	●	●	10	GPIO15	GPIO15	low
low	GPIO17	GPIO17	11	●	●	12	GPIO18	GPIO18	low
low	GPIO27	GPIO27	13	●	●	14	GND	DNC	
low	GPIO22	GPIO22	15	●	●	16	GPIO23	GPIO23	low
	DNC	+3,3 V	17	●	●	18	GPIO24	GPIO24	low
low	GPIO10	GPIO10	19	●	●	20	GND	DNC	
low	GPIO9	GPIO9	21	●	●	22	GPIO25	GPIO25	low
low	GPIO11	GPIO11	23	●	●	24	GPIO8	GPIO8	high
	DNC	GND	25	●	●	26	GPIO7	GPIO7	high

+3,3 V	+3,3 Volt
+5 V	+5 Volt
GND	ground
DNC	do not connect
GPIO	general purpose input output

Tabelle 1: Pin-Belegung der 26-poligen Stiftleiste P1 des Raspberry Pi

Solltet ihr ein altes Board der Rev. 1 haben, dürft ihr die mit „DNC“ gekennzeichneten Pins nicht verwenden. Außerdem sind GPIO0 und GPIO1 in Rev. 2 jetzt GPIO2 und GPIO3.

Die +3,3 Volt dürfen höchstens mit 50 mA belastet werden. Die +5 V hängen direkt mit der Spannungsversorgung des Raspberry Pi zusammen und liefern so viel Strom, wie unser angeschlossenes Netzteil her gibt. Wir sollten es aber nicht übertreiben, sonst brennen die dünnen Leiterbahnen auf der Platine ab. Die GPIO Ausgänge schalten zwischen 0 V und +3,3 V und sind mit maximal 16 mA belastbar[154].

9.1 Die Software zum GPIO Stecker

Um die Stiftleiste mit einem c/c++ Programm steuern zu können, verwenden wir ein Bibliothek, die wir aus dem Internet bekommen[155]. An dieser Stelle wollen wir uns bei Mike MC Cauley bedanken, für die gute Arbeit und dafür, dass er die Bibliothek als Open Source unter GPL V2 lizenziert. Um die Bibliothek zu installieren gehen wir mit dem Midori Webbrowser auf die Webseite:

http://www.open.com.au/mikem/bcm2835/index.html

Die aktuelle Version zur Zeit, als ich dieses Buch schrieb war die Version 1.17. und wir können die

154 Achtung! Die GPIOs sind direkt mit dem Broadcom Mikro-Prozessor BCM2835 verbunden. Kurzschlüsse am GPIO können den Prozessor beschädigen.
155 Ihr findet sie auch auf der beiliegenden CD.

komprimierte Installationsdatei direkt aus dem Internet laden indem wir auf den link:

http://www.open.com.au/mikem/bcm2835/bcm2835-1.17.tar.gz

klicken. Alternativ können wir uns die Datei mit dem PC von der CD auf einen USB-Stick kopieren. Wenn wir den USB-Stick dann in den Raspberry Pi stecken, können wir mit dem File-Manager eine Kopie in unser Homeverzeichnis /home/pi kopieren. Hier geben wir auf der Kommandoebene folgendes ein:

```
tar zxvf bcm2835-1.17.tar.gz
cd bcm2835-1.17
./configure
make
sudo make check
sudo make installation
```

Wem das zu schnell ging und erklärt haben will, was hier abläuft ließt sich im nachfolgenden Absatz durch was Sache ist. Wen das eher langweilt, kann schon zur nächsten Überschrift.

Das entpacken der Datei machen wir auf der Kommandozeile mit dem Befehl „tar“. Anschießend wechseln wir mit cd in das neue Verzeichnis bcm2835-1.17. Alternativ können wir natürlich auch durch doppelklicken im File-Manager auf die Datei bcm2835-1.17.tar.gz die Datei entpacken. Es öffnet sich der Xarchiver ,wie er in Bild 19 zu sehen ist. Durch klicken auf den Filenamen zeigt er uns an, was in der bcm2835-1.17.tar.gz so alles drin ist, wenn wir .

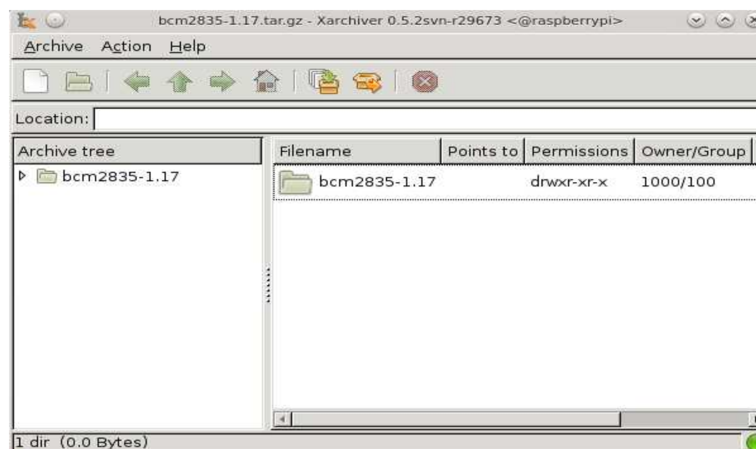


Abbildung 19: Der Xarchiver zum entpacken von komprimierten Dateien

Klicken wir oben in der Zeile auf das Symbol mit dem Pappkarton und dem orangen Pfeil, speichert sich das Verzeichnis bcm2835-1.17 in unser Homeverzeichnis /home/pi. Das Verzeichnis können wir uns jetzt im File-Manager anschauen und den Xarchiver wieder schließen. Es gibt eine ganze Menge folder und files. Zu den foldern gehören die Verzeichnisse: doc, example und scr[156]. Wir schauen uns alles mal an, verändern aber nichts. Unter den files gibt es eine README[157] Datei, die

156 Das sind die Verzeichnisse in denen die Dokumentation, Beispiele und Sourcecodes stehen.
157 Es ist guter Stiel, und den beherrscht Mike natürlich, in unseren Projekten einen Textfile mit dem Namen README

wir uns auch mal durchlesen und notfalls mit Hilfe der Webseite http://www.leo.org übersetzen. Wir finden auch einige Makefiles und configure Dateien. Eine der configure-Datei hat keine extension und ist ausführbar.

Übrigens, wenn wir uns im File-Manager befinden und die Taste F4 drücken, öffnet die ein Kommandozeilen-Fenster mit der bash und wir sind schon direkt in dem richtigen Verzeichnis. Das erkennen wir natürlich sofort am Prompt der jetzt „pi@raspberrypi ~/bcm2835-1.17 $“ heißt. Sollte uns der schwarze Hintergrund im Kommandozeilen-Fenster stören, können wir durch klicken auf Edit und dann auf Prefrences die Hintergrund[158] Farbe verändern. Mit „ls“ können wir uns wieder anzeigen was im Verzeichnis drin steht. Auf der Kommandozeile starten wir jetzt das configure Programm mit dem Befehl:

./configure

Es folgen jede Menge Informationen, die die Abhängigkeiten mit anderen Installationspaketen überprüfen und bei Erfolg die Installation durchführen. Jetzt müssen wir noch den richtigen Makefile mit dem Programm „make“ aufrufen mit dem Befehl:

make

Anschließend lassen wir noch von dem Programm „make“ alles überprüfen mit dem Befehl:

sudo make check

Ihr erkennt an dem vorangestellten „sudo“, dass wir uns zuvor root-Rechte besorgt haben. Diese brauchen wir auch, um die Bibliotek endgültig zu installieren mit dem Befehl:

sudo make install

Jetzt sind wir gut vorbereitet um ein erstes kleines c/c++ Programm zu schreiben, mit der wir eine am GPIO angeschlossene LED blinken lassen können.

9.2 Wir lassen LEDs blinken

Eigentlich wollen wir ja Servos ansteuern, aber es ist ganz gut, wenn wir erst mal mit was Einfachem lernen, wie die GPIOs angesteuert werden. Wenn wir an den GPIOs Leuchtdioden blinken lassen können, ist das Ansteuern von Servos auch nicht mehr so schwer. Beginnen wir also mit etwas Hardware. Die GPIO Pins am Raspberry Pi können, wenn sie als digitale Ausgänge programmiert sind, Spannungen von 0V und +3,3 V ausgeben. 0 V entspricht einer logischen 0 (Null oder „low“), 3,3 V entspricht einer logischen 1 („high“).

Um eine LED leuchten zu lassen, beschäftigen wir uns etwas mit Elektrotechnik, aber keine Angst, es reicht das ohm'sche[159] Gesetz. Was Ströme und Spannungen sind, ist erst mal schwer zu fassen, da wir diese weder sehen, anfassen noch spüren können[160]. Vielleicht hilft euch meine Analogie zur Wasserleitung. Der Druck in der Wasserleitung entspricht der elektrischen Spannung.

---

oder LIESMICH aufzunehmen und darin Informationen zum Projekt zu speichern.

158 „Hintergrund“ heißt aus englisch „background“.

159 Das Ohm'sche Gesetz ist nach Georg Simon Ohm benannt. Es besagt, das Strom und Spannung umgekehrt proportional zueinander sind und die Proportionalität der Widerstand des elektrischen Leiters ist. Außerdem ist „Ohm“ auch die Einheit des elektrischen Widerstands, die mit dem griechischen Buchstaben Omega „Ω“ abgekürzt wird.

160 …. es sei denn, wir greifen in die Steckdose. Wir haben es hier aber nur mit Spannungen von maximal 5 V zu tun, und das ist vollkommen ungefährlich. Wer Netzteile aufschraubt, lebt hingegen gefährlich und sollte zumindest wissen was er tut.

Spannungen[161] werden in Volt[162] gemessen. Die Wassermenge, die durch die Leitung fließt, ist der Strom[163], den wir in Ampere[164] messen.

Wenn ein GPIO Ausgang auf „high“ programmiert ist, sollt ein Strom durch die Leuchtdiode[165] fließen. Die LED fängt dann an zu leuchten. Wir schauen uns dazu den Schaltplan aus Bild 20 an.

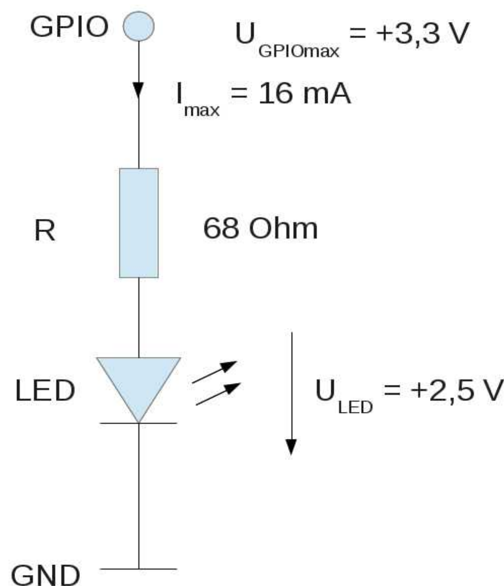


Abbildung 20: Schaltplan zur Ansteuerung einer LED an einem GPIO Pin.

In Bild 12 sehen wir von oben nach unten mehrere Symbole. Der oberste Kreis soll den GPIO Pin symbolisieren. Von dort geht ein Anschluss an den Widerstand. Widerstände werden in Europa als Rechteck[166] gezeichnet und international als Formelzeichen mit R[167] bezeichnet. Auf der Verbindungslinie, die eine elektrisch leitende Verbindung[168] symbolisiert, ist ein kleiner Pfeil gezeichnet. Dieser ist das Symbol für den Strom, und gibt die technische Stromrichtung an. Ströme haben das Formelzeichen I. Aus dem Widerstand R fließt der Strom in die LED. Das Symbol einer LED ist das Gleiche wie das einer Diode; ein Dreieck, dessen Spitze die technische Stromrichtung

161 „Spannung“ heißt auf englisch „voltage“. Das Formelzeichen ist „U“ und im englischen „V“. Die Einheit Volt wird immer mit „V“ abgekürzt.

162 „Volt“ wurde nach dem Italiener Alessandro Volta im vorletzten Jahrhundert so benannt. Allesandro hatte schon immer eine Leidenschaft für Sümpfe und die richtige Erklärung für die animalische Elektrizität, die sein Freund von Luigi Galvani an toten Fröschen entdeckt hatte. Heute würde den beiden sicherlich der Tierschutzverein auf die Pelle rücken. Damals wurden sie von Napoleon höchst persönlich belobigt. Es gab sogar Orden.

163 Das Formelzeichen für Strom ist „I“, die Einheit wird mit „A“ abgekürzt.

164 Die elektrische Stromstärke messen wir in „Ampere“, abgekürzt A. Benannt wurde diese Einheit nach dem Franzosen Andre-Marie Ampere, der ein absolutes Multitalent war. Genial war damals seine Erkenntnis, dass elektrische Ströme Magnetfelder verursachen. Andre-Marie konstruierte ein Messgerät zur Messung von Strom, das er Galvanometer nannte. Wisst ihr noch wie Alessandro's Freund Luigi mit Nachnamen hieß? In Ermangelung besseren Wissen unterlief Andre-Marie ein folgenschwerer Fehler, an dem die Physiker heute knabbern. Andre definierte die technische Stromrichtung in entgegengesetzter Richtung zur Flussrichtung der Elektronen. Damals war das Elektron leider noch nicht entdeckt, und wir müssen Ampere diesen Lapsus verzeihen.

165 Leuchtdiode ist das Selbe wie LED. „LED“ steht für „light emiting diode“, also Licht aussendende Diode.

166 In den USA zeichnet man eine Zackenlinie.

167 „R“ steht für englisch „resistor“, auf deutsch Widerstand.

168 In unserem Fall ist das ein Draht.

angibt mit einem waagerechten Strich an der Spitze. An die LED werden nur noch zwei kleine Pfeile dran gemalt. Diese symbolisieren das emittierte Licht. Der Strom, der aus der LED fließt, fließt gegen GND ab. Der Strom durch den Widerstand ist also der gleiche, wie durch die LED. Wo soll er auch sonst hin. Wenn wir uns das Datenblatt einer LED anschauen, werden wir feststellen, dass so ziemlich alle roten, gelben und grünen LEDs in Durchlassrichtung einen Spannungsabfall[169] von etwa 2.5 V haben. Wenn hingegen der GPIO Pin auf „low“ liegt, also bei 0 Volt, kann auch kein Strom fließen[170] und ohne Strom leuchtet die LED nicht und es gibt auch keinen Spannungsabfall. Bei +3,3 V am GPIO muss also die restliche Spannung am Widerstand abfallen. An dem Widerstand ist der Spannungsabfall dann also:

$U_R = U_{GPIO} – U_{LED} = 3{,}3\ V – 2{,}5\ V = 0{,}8\ V$

Wir wissen vom Raspberry Pi, dass der Strom, der aus dem GPIO fließt, nicht größer als 16 mA[171] sein darf, da er sonst kaputt zu gehen droht. Mit dem ohm'schen Gesetz können wir also jetzt die minimale Widerstandsgröße berechnen:

$R = U_R / I = 0{,}8\ V / 16\ mA = 50\ Ohm$

Der Widerstand sollte also größer als 50 Ohm sein, dann ist der Strom, der aus dem GPIO fließt kleiner als 16 mA. Mit einen Widerstand von 68 Ohm, sind wir also auf der sicheren Seite.

9.2.1 Die Hardware

Jetzt müssen wir das ganze noch aufbauen. Für euch habe ich einen typischen Widerstand in Bild 21 und eine typische LED in Bild 22 fotografiert.



Abbildung 21: Ein elektronischer Widerstand

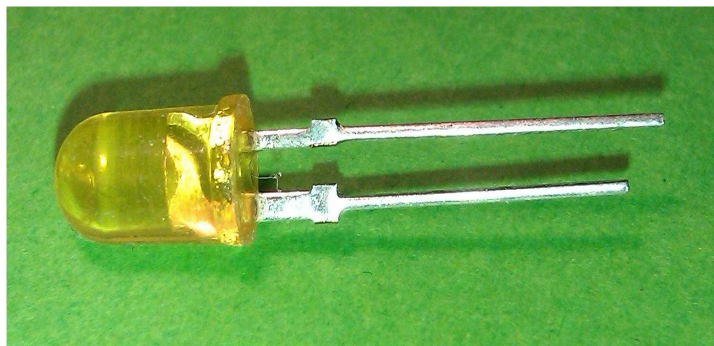


Abbildung 22: Eine Leuchtdiode (LED)

Widerstände und LEDs gibt es in verschiedenen Bauformen. Wir können am Besten die bedrahteten Bauteile gebrauchen. Widerstände haben zwei Anschlüsse, die gleichwertig sind. Es ist vollkommen egal, wie rum ich ihn anschließe. Bei der LED, oder allgemein bei Dioden, ist das nicht so. Der Strom kann nur von der Anode zur Kathode fließen, so heißen die Anschlüsse einer Diode. In die andere Richtung sperrt die Diode, das heißt sie lässt keinen Strom fließen und ohne Strom

169 Der Spannungsabfall ist nichts für die Mülltonne. Im Sprachgebrauch von Elektronikern ist hier gemeint, das diese Spannung an den beiden Klemmen des Bauelements anliegt, wenn ein Strom dadurch fließt.

170 Denn GND liegt auch auf 0V.

171 „mA“ steht für Milliampere. 16 mA = 0,016 A, oder 10^{-3} A. Das kleine m vor einer Einheit erspart uns die Null vor und nach dem Komma und auch noch das Komma selbst. Naturwissenschaftler und Ingenieure finden diese Schreibweise unglaublich praktisch und so gibt es auch noch µ, n, f (micro, nano, femto) für die sehr kleine Größen (10^{-6}, 10^{-9}, 10^{-12}), aber auch das k, M, G (kilo, Mega, Giga) für die Großen (k = 1000=10^3, 10^6, 10^9)

leuchteten LEDs auch nicht. Leuchtdioden brauchen also eine Kennzeichnung, damit wir sehen, wo die Kathode ist. Wenn ich eine neue LED habe, erkenne ich das an dem kürzeren Drahtbeinchen[172]. Da fließt also der technische Strom aus der LED wieder raus und in unserem Falle direkt zum GND.

Man kann die Bauteile natürlich zusammen löten. Ich habe mir da im Elektronik Handel so einen Stiftleisten Stecker besorgt und die Bauteile auf einem Steckbrett miteinander verbunden. Außerdem habe ich gleich drei LEDs mit Widerständen mit den GPIO Pins GPIO10, GPIO9 und GPIO11 verbunden. Das sind die Pins 19, 21 und 23. Der Pin 25 ist GND. Die anderen Kabel hängen einfach in der Luft. Meinen Aufbau könnt ihr in Bild 23 bewundern.

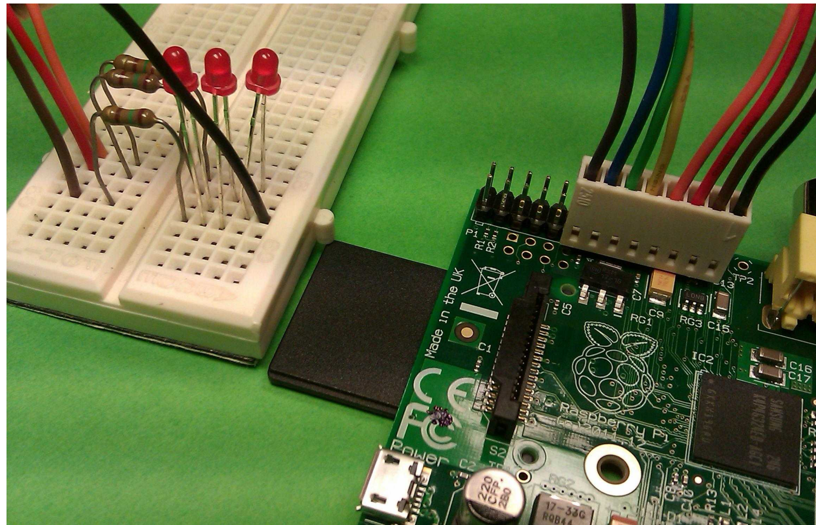


Abbildung 23: Drei LEDs mit Widerständen an der Stiftleiste P1 angeschlossen.

9.2.2 Die Software

Jetzt muss das Ganze noch über ein Programm angesteuert werden. Dazu gehen wir von Mike's Beispiel blink.c aus, das wir im Verzeichnis /home/pi/bcm2835-1.17/examples finden. Wir öffnen es mit dem Emacs und speichern es gleich wieder unter einem neuen Namen und in einem neuen Verzeichnis, indem wir auf „File“ und dann „Save as ..“ klicken. Wir wechseln jetzt zu dem Verzeichnis „c++Progs“ und legen ein neues Verzeichnis mit dem Namen „ZweitesProjekt“ an. Der Datei geben wir jetzt den Namen „dreiLEDsblinken.cpp“. Wenn ihr wollt, könnt ihr natürlich auch auf die Kommandoebene und mit den Befehlen cd, ls, mkdir und cp das selbe Ergebnis erzielen. Wenn ihr es nicht auf Anhieb schafft, seid ihr in bester Gesellschaft. Üben und ausprobieren sind der Schlüssel zum Erfolg. Ich koch mir, wenn' klappt, dann immer eine Tasse Kaffee zur Belohnung. Bei Misserfolgen spüle ich den Frust mit dem Kaffee runter und probiere es später nochmal.

Wir ändern jetzt erst mal die erste Kommentar-Zeile von Mike's Programm in „dreiLEDsblinken.cpp“ ab und lesen uns durch, was Mike uns so geschrieben hat. Mike schreibt uns zum Beispiel, wie wir das Programm mit dem gcc compilieren sollen, und das wir es dann mit dem voran gestellten Befehl „sudo“ starten sollen. Das merken wir uns für den Makefile. Unter

172 Ich habe so eine Eselsbrücke. Das kürzere Bein ist abgeschnitten, zu Englisch „cut“. „Kathode“ heißt auf englisch „cathode“. Merkt ihr's? Naja, es ist halt eine Eselsbrücke.

Author: schreiben wir jetzt unseren Namen dazu ohne den Hinweis auf Mike zu löschen. Wir können uns auch noch beim Urheberrecht[173] erwähnen. Wir wissen, das Mike seine Programme unter GPL gestellt hat. Hier hat er es nicht extra erwähnt. Wir holen das nach. Mein Kommentar-Text sieht dann aus wie in Bild 24.

emacs@linux-blai <2>

File Edit Options Buffers Tools C++ Help

Save Undo

```
// dreiLEDsblinken.cpp
//
// Example program for bcm2835 library
// Blinks a pin on an off every 0.5 secs
//
// After installing bcm2835, you can build this
// with something like:
// gcc -o blink -l rt blink.c -l bcm2835
// sudo ./blink
//
// Or you can test it before installing with:
// gcc -o blink -l rt -I ../../src ../../src/bcm2835.c blink.c
// sudo ./blink
//
// Author: Mike McCauley (mikem@open.com.au) mit Ergänzungen durch Axel Hülsmann
// Copyright (C) 2011 Mike McCauley und (C) 2012 Axel Hülsmann
// $Id: RF22.h,v 1.21 2012/05/30 01:51:25 mikem Exp $
// Lizenz: GPL v3.
// Dies Programm ist freie Software. Du kannst es unter den Bedingungen der GNU
// General Public License, wie von der Free Software Foundation veröffentlicht,
// weiter geben und/oder modifizieren. Die Veröffentlichung dieses Programms erfolgt
// ohne irgendeine Garantie.
// Die GPL kannst du unter http://www.gnu.org/lisences nachlesen

#include <bcm2835.h>

// Blinks on RPi pins GPIO 19, 21 und 23
#define LED1 RPI_GPIO_P1_19
#define LED2 RPI_GPIO_P1_21
#define LED3 RPI_GPIO_P1_23
```

Abbildung 24: Der erste Teil der Datei dreiLEDsblinken.cpp im Emacs

Die erste Zeile nach dem Kommentar-Block ist eine „include[174]“ Anweisung an den Compiler. Anweisungen an den Compiler werden in c/c++ immer mit einem „#“ eingeleitet. Es soll damit die Datei bcm2835.h eingefügt werden.

An dieser Stelle müssen wir uns kurz mit Header-Dateien[175] beschäftigen, um zu verstehen was das soll. Der Sinn dahinter ist, das wir das Hauptprogramm mit neuen Funktionen bekannt machen wollen, die in der Header-Datei definiert sind. Zu jeder Header-Datei gehört aber auch eine c/c++

173 Unser europäisches Urheberrecht ist vergleichbar mit dem Copyright der USA. Wir Europäer brauchen aber ein Copyright nicht extra erwähnen, da wir das Urheberrecht als Autor auch ohne Hinweis haben, denn es ist bei uns nicht veräußerbar. Falls gegen die Lizenz aber in den USA verstoßen wird, wovon wir bei diesem Programm nicht ausgehen, lässt sich unser Recht besser durchsetzen.

174 „include“ heißt „einfügen“

175 Header-Dateien enthalten c/c++ Sourcecode, dessen Anweisungen zuerst compiliert wird. Wir hatten das schon mal im ErstenProjekt mit der stdio.h.

Datei mit gleichem Namen aber der Endung „c“ oder „cpp“, in der die Arbeitsweise der Funktion beschrieben ist. Meistens interessiert uns das gar nicht so genau. Wir wollen nur wissen, was die neuen Funktionen können. Wie sie das machen, überlassen wir anderen. Die Header-Datei bcm2835.h von Mike definiert einige ganz nette Funktionen, mit denen wir nicht nur die GPIOs steuern können. Diese schauen wir uns noch im Detail an.

Es folgen einige „define[176]“ Anweisungen an den Compiler. Define Anweisungen sind leicht zu verstehen. Anstelle eines langen Bandwurms, hier zum Beispiel „RPI_GPIO_P1_09“[177] können wir jetzt im Sourcecode kurz und prägnant „LED1“ schreiben.

176 „define“ heißt „definiere“.

177 Wenn wir einen neueren Raspberry Pi der Revision 2 haben, sollten wir hier statt RPI_GPIO_P1_xx RPI_V2_GPIO_P1_xx schreiben, da sich die Pin Belegung inzwischen geändert hat. „xx“ steht natürlich für die Pin Nummer an P1. Schaut euch dazu Tabelle 1 genauer an.

```
emacs@linux-blai
File  Edit  Options  Buffers  Tools  C++  Help
Save  Undo

int main(int argc, char **argv)
{
    if (!bcm2835_init())
        return 1;

    // Set the pin to be an output
    bcm2835_gpio_fsel(LED1, BCM2835_GPIO_FSEL_OUTP);
    bcm2835_gpio_fsel(LED2, BCM2835_GPIO_FSEL_OUTP);
    bcm2835_gpio_fsel(LED3, BCM2835_GPIO_FSEL_OUTP);

    // Blink
    while (1)
    {
        // Turn it on
        bcm2835_gpio_write(LED1, HIGH);
        // wait a bit
        bcm2835_delay(500);
        // Turn it on
        bcm2835_gpio_write(LED2, HIGH);
        // wait a bit
        bcm2835_delay(500);
        // Turn it on
        bcm2835_gpio_write(LED3, HIGH);
        // wait a bit
        bcm2835_delay(500);
        // turn it off
        bcm2835_gpio_write(LED1, LOW);
        // wait a bit
        bcm2835_delay(500);
        // turn it off
        bcm2835_gpio_write(LED2, LOW);
        // wait a bit
        bcm2835_delay(500);
        // turn it off
        bcm2835_gpio_write(LED3, LOW);
        // wait a bit
        bcm2835_delay(500);
    }

    return 0;
}
U:**-  dreiLEDsblinken.cpp   Bot L36    (C++/l Abbrev)
CRM Buffer                   Size  Mode              File
 %  *GNU Emacs*               684  Fundamental

U:%%-  *Buffer List*   Bot L5     (Buffer Menu)
Switch to buffer (default *scratch*):
```

Abbildung 25: Das Hauptproramm von dreiLEDsblinken im Emacs

Jetzt folgt das Hauptprogramm. Mike hat hier den vollständigen Aufruf der main-Funktion beschrieben. Das wäre nicht nötig gewesen, ist aber guter Stil. Wir sehen das oben in Bild 25 . Wenn Funktionen ein Ergebnis haben, muss natürlich auch festgelegt werden, welchen Typ dieser

hat. Der Rückgabewert[178] jeder Funktion wird mit dem Befehl „return“ übergeben und ist bei unserer main-Funktion eine ganze Zahl und daher vom Typ „Integer“. Deshalb steht hier auch vor „main“ die Typenbezeichnung „int“. Der Emacs hat erkannt, das „int“ kein Befehl ist, sondern ein Typ und das Wort grün eingefärbt. Das Wort „main“ hat er als Hauptfunktion erkannt und blau eingefärbt. Bei der main-Funktion hat man sich darauf geeinigt, das, wenn sie fehlerfrei durchläuft, eine „0“ zurück liefern soll. Sollten andere Zahlen erscheinen, weiß man, das etwas falsch gelaufen ist. Der main-Funktion werden beim Aufruf die Parameter „argc“, das ist wieder eine ganze Zahl vom Typ „int“, und „**argv“ vom Typ „char[179]“ übergeben. Damit hat es folgendes auf sich. Rufen wir ein Programm von der Kommandozeile auf, könnten wir ihm Parameter übergeben. Diese Parameter kann dann das Programm bearbeiten. Wir brauchen das hier aber nicht und werden uns momentan nicht weiter damit belasten[180].

Die erste Anweisung im main-Programm ist eine if[181]-Anweisung. Wenn genau das passiert, was in der runden Klammer hinter „if“ steht, dann soll das, was dahinter steht ausgeführt werden. Falls nicht, wird der hintere Teil ignoriert. Was in der runden Klammer steht, muss entweder wahr sein, oder falsch[182]. Die if-Anweisung steht hier in zwei Zeilen und wird erst dann mit dem obligatorischen „;“ beendet. Diese zwei Zeilen dienen also nur der besseren Lesbarkeit. Eine Zeile hätte es auch getan. Das erste Zeichen nach der runden Klammer ist ein Ausrufezeichen „!“. Dies steht für eine logische Verneinung. Hinter dem „!“ wird die Funktion „bcm2835_init()“ aufgerufen. Diese Funktion wurde in der Header-Datei definiert[183]. Sie initialisiert[184] die GPIO-Schnittstelle[185]. Sollte diese Funktion einen Fehler feststellen, liefert sie keine logische „0“ zurück. „!0“ ist aber das Gegenteil von falsch, also wahr „1“. Somit wird auch der Befehl hinter der if-Anweisung ausgeführt. Da steht aber „return“, der Befehl, der die main-Funktion sofort beendet und eine „1“ übergibt, was ja soviel heißt wie Fehler, wie wir inzwischen wissen! Ganz schön verschachtelt das Ganze, oder? Blickt ihr noch durch? Diese if-Anweisung macht also zwei Dinge gleichzeitig.

Erstens: Sie initialisiert die GPIO-Schnittstelle.

Zweitens: Sie stellt fest, ob beim Initialisieren ein Fehler auftritt und teilt das dann dem Hauptprogramm mit.

Ja, das ist schon gehobener c-Programmierstil. Den werden wir uns natürlich merken und später selbst anwenden.

Es folgen drei Funktionsaufrufe der Funktion „bcm2835_gpio_fsel(...)“. Diese Funktion wurde wieder im Header-File „bcm2835.h“ definiert und sagt der GPIO-Schnittstelle welche Pins Ausgänge sind. Per default, also wenn nicht anderes mitgeteilt wird, sind die GPIO-Pins nämlich Eingänge. Das ist übliche Praxis bei Microcontrollern, denn so ist sichergestellt, das nicht irgendwelche Spannungen anliegen die zufällig kurzgeschlossen werden und den Broadcom Microprozessor zerstören könnten. Das erste Argument in der runden Klammer von „bcm2835_gpio_fsel(...)“ haben wir mit „#define“ anfangs umdefiniert. Es gibt an, welche Pin-Nummer angesteuert werden soll. Das zweite Argument ist die Festlegung, das dieser Pin jetzt ein

178 Der Rückgabewert ist das Ergebnis der Funktion.
179 „char“ steht für „character“. Das mit sind in erster Line Buchstaben, aber auch Sonderzeichen oder Ziffern gemeint.
180 Wer es genauer wissen will kann sich im Interget schlau machen oder die angegebene Literatur studieren.
181 „if“ steht für „wenn“.
182 Wahr entspricht „1“, falsch entspricht „0“.
183 Schaut euch die Datei bcm2835.h und bcm2835.c in aller Ruhe mal an und versucht zu verstehen was da abgeht. Ganz schön kompliziert, oder? Gut, das wir die nicht selbst schreiben mussten!
184 Unter initialisieren versteht man, alles in einen geordneten Zustand zu bringen.
185 Eine Schnittstelle, auf englisch „interface“ ist eine Verbindung zwischen zwei „Welten“. Hier zwischen dem Raspberry Pi und unserer Roboter-Ameise.

Ausgang ist.

Bis jetzt hat sich noch nicht viel in unserem Programm getan. Alles war nur notwendige Vorbereitung. Der eigentliche Teil passiert erst in der nachfolgenden while-Schleife[186]. Alles was in den geschweiten Klammern der while-Schleife steht, wird so lange ausgeführt, bis das, was zwischen den runden Klammer nach „while“ steht, falsch wird, also logisch „0“. Bei dieser while-Schleife wird das jedoch nie passieren, denn zwischen den runden Klammern steht explizit „1“ und das ist nun mal nicht „0“, also nicht falsch. Alles was zwischen den geschweiften Klammern steht wird so bis zum St. Nimmerleinstag ausgeführt[187].

Als erstes kommt in der while-Scheife die Funktion bcm2835_gpio_write(...). Diese ist auch in Mike's Header-Datei definiert worden und legt an den Pin, der als erstes Argument der Funktion übergeben wird auf „HIGH“ (dann leuchtet die angeschlossene LED), oder auf LOW (dann geht sie aus). Die Nächste Funktion „bcm2835_delay(...)“, gibt einfach nur an, wie lange der Raspberry Pi nichts tun soll. Die Zahl, die als Argument übergeben wird, gibt die Millisekunden an, die vertrödelt werden soll.

Ja, das war's auch schon. Eigentlich ganz einfach, oder? Nacheinander werden die drei LEDs angeschaltet und dann wieder aus. Dann geht’s wieder von vorn los uns immer so weiter, bis wir Ctrl-C drücken ('tschuldigung Strg C).

9.2.3 Das Programm compilieren und ausführen

Es gibt selten ein Programm, das auf Anhieb läuft. Der Emacs hilft uns ein bisschen, Fehler rechtzeitig zu erkennen, indem er alles, was er kennt, farbig kodiert. Übrigens, der häufigste Anfängerfehler bei c/c++ ist das vergessen von „;“ am Ende einer Befehlszeile.

Wir wollen unser Programm jetzt compilieren. Dazu erstellen wir noch mit dem Emacs einen Makefile in unsrem Projektordner „ZweitesProjekt“.

```
# Makefile für dreiLEDS

dreiLEDS: dreiLEDsblinken.c
	gcc -o dreiLEDS -l rt dreiLEDsblinken.c -l bcm2835
```

Jetzt können wir im Emacs unter Tools wieder Compile .. anklicken und mit der Bestätigung der aufkommenden Fragen das Programm „dreiLEDS“ compilieren. Wir schauen uns die Meldungen genau an. Es sollten jetzt kein Fehler aufgetreten sein, sonst müssen wir noch mal dran. Da das Programm jetzt Hardware anspricht, brauchen wir root-Rechte um es zu starten und geben daher ein:

sudo ./dreiLEDS

Wir haben es geschafft. Nacheinander brennen die drei Leuchtdioden und gehen dann nacheinander auch wieder aus. Herzlich Glückwunsch zu diesem Erfolg!

---

186 „while“ steht hier für „bis“.

187 Oder bis wir den Strom-Stecker ziehen. Den Stecker ziehen gilt unter Hackern als „unsportlich“. Wir können es auch mit Ctrl-C versuchen. Ctrl-C steht für „control C“, also das gleichzeitige drücken der „Strg“ und der „c“ Taste. „Steuerung“ heißt auf englisch „controller“

10 Servos

An dieser Stelle beschäftigen wir uns ein bisschen mit Servos[188] aus dem Modellbau, mit denen wir die Beine unserer Roboter-Ameise bewegen. Servos sind richtige elektromechanische Wunderwerke, die es inzwischen für wenig Geld gibt. Sie bestehen aus einem Motor, einem Getriebe und einer Ansteuerungselektronik. Ein billiges, in die Roboter-Ameise eingebautes Servo sehen wir in Bild 26 .

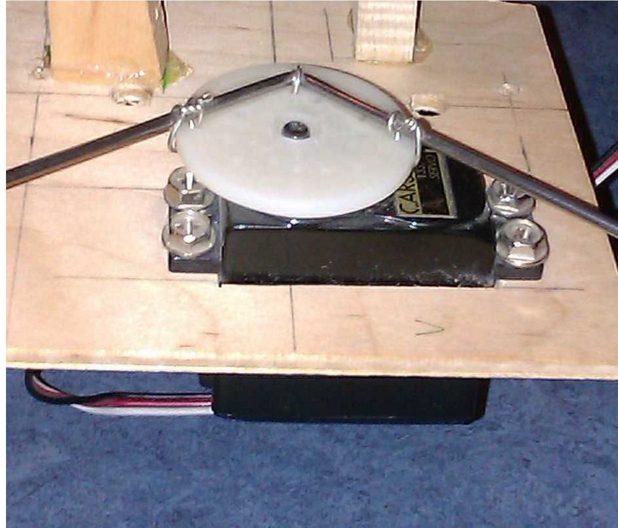


Abbildung 26: Ein eingebautes Servo

Dieses Servo hat Abmessungen von ca. 40x40x20 mm³. Mit 4 Schrauben kann man es in der Aussparung eines Sperrholzbrettchens befestigen. Oben hat es eine Scheibe, die sich nach Rechts und Links drehen kann, auf der wir einen gebogenen Draht befestigen können und damit gleich zwei Beinchen gleichzeitig bewegen können.

10.1 Wie funktioniert ein Servo?

Ein Motor treibt über ein Getriebe die Scheibe an. Die Drehposition wird über ein internes, synchron mitlaufendes Dreh-Potentiometer[189] abgefragt. Dieses fungiert als Winkelmesser der Drehscheibe. Bild 27 zeigt ein Blockschaltbild eines Servos.

188 Modellbau Servos werden seltener auch als Rudermaschine bezeichnet. Servos gehören zu den so genannten „Aktuatoren“. Aktuatoren sind elektromechanische Bauelemente.
189 Ein Potentiometer ist ein verstellbarer Widerstand mit drei Anschlüssen.

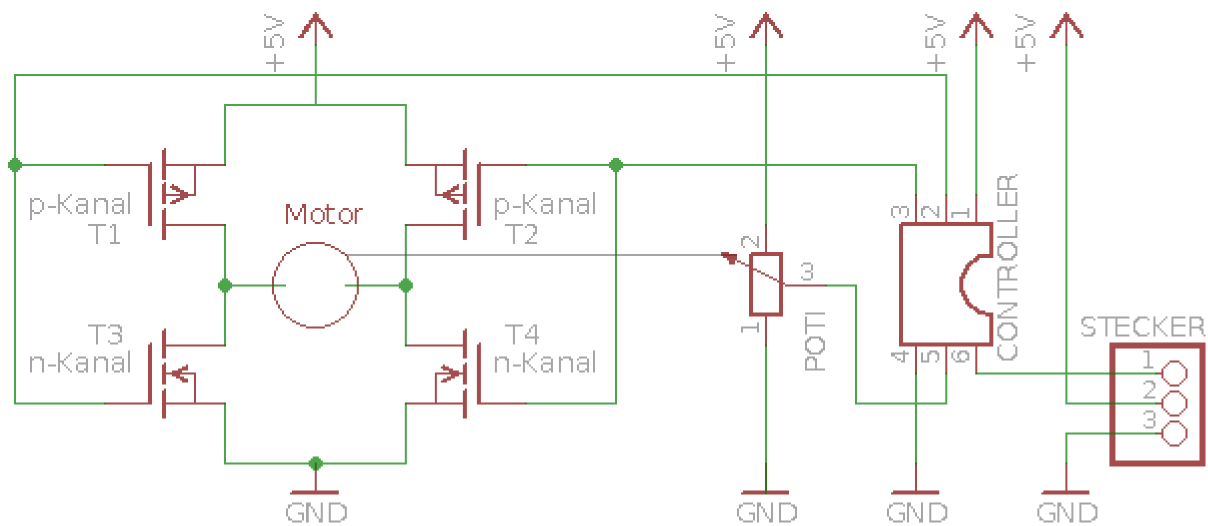


Abbildung 27: Schaltplan eines Servos

In Bild 27 habe ich euch die Ansteuerung des Motors mit Feldeffekt-Transistoren eingezeichnet. Soll der Motor rechts herum drehen, leiten die Transistoren T1 und T4, während T2 und T3 sperren. Das wird erreicht indem die Steuerelektrode[190] der FETs[191] T1 und T3 vom Controller auf GND und von T2 und T4 auf +5V geschaltet werden. Sind alle Gates auf 0 V oder +5V, kann kein Strom durch den Motor fließen.

Legt man an das Potentiometer eine Spannung an, so kann man am Mittelabgriff eine Spannung messen, die Proportional der Verstellung ist. Diese gemessene Winkelstellung wird ständig mit dem Soll-Winkel verglichen. Stimmen Mess- und Soll-Winkel überein, wird der Motor ausgeschaltet. Ist die Differenz zwischen Soll- und Ist-Winkel zu groß wird der Motor unter Spannung gesetzt. Ist die Differenz zu klein, wird die Spannung am Motor umgepolt. Die gewünschte Winkelstellung wird dem internen Controller IC nicht durch eine analoge Spannung mitgeteilt, sondern durch ein PWM[192]-Signal über den Stecker, wie er in Bild 28 zu sehen ist.

---

190 Die Steuerelektroden eines FETs nennt man „Gates“. Das hat jetzt aber wirklich nichts mit dem Gründer der Firma Microsoft zu tun. Die Elektrode, die an GND oder der Spannungsquelle angeschlossen wird heißt „Source“, von englisch „Quelle“. Die dritte Elektrode, der Strom geschaltet wird heißt „Drain“, von englisch „Senke“.

191 FET ist die Abkürzung für Feld-Effekt-Transistor. FETs sind, vereinfacht ausgedrückt, spannungs-gesteuerte Stromschalter.

192 PWM steht für Puls-Weiten Modulation.

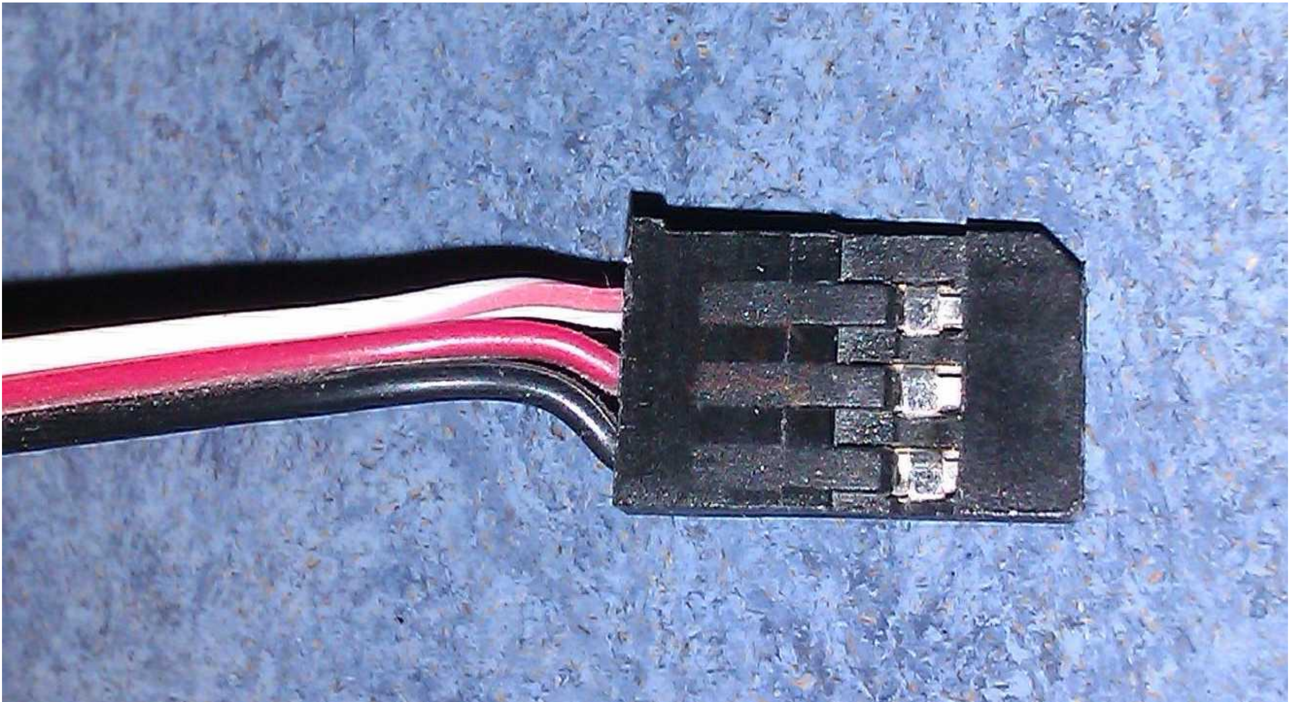


Abbildung 28: Ein typischer Verbindungsstecker eines Servos

Ein Servo-Stecker hat daher immer drei Anschlüsse: GND, +5V und das PWM-Signal[193]. Wenn ihr neue Servos kauft, achtet darauf was sie für einen Stecker haben. Ich empfehle euch Futuba-, Graupner- oder Conrad-Stecker. Die haben einen Kontakt-Abstand vor 2,54 mm[194] und passen auf die normalen Stiftleisten.

Schauen wir uns noch an, wie das PWM-Signal eines Servos aussehen muss. Mit der Weite eines Pulses wird der Winkel der Drehscheibe definiert. Dieser Puls wird mit einer Frequenz von 50 Hz wiederholt. 50 Hz entspricht einer Periodendauer von 20 ms. Die Periodendauer ist unkritisch und muss nicht so genau eingehalten werden. Die Pulsweite selbst liegt zwischen 1 ms (linker Anschlag) und 2 ms (rechter Anschlag). Während des Pulses liegt die Spannung auf High-Pegel[195] und den Rest der Periodenlänge auf Low-Pegel[196] ist. In Bild 29 habe ich euch ein schematisches Zeitdiagramm gemalt.

193 Meistens sind die drei Leitungen des Servos noch farblich kodiert, wie in dem Bild gezeigt. GND ist schwarz, +5V ist rot und das PWM- Signal ist gelb oder weiß.

194 1 Zoll = 1 inch = 25,4 mm. Ein zehntel inch sind 1/10 Zoll, also 2,54 mm. Und dieses Maß lieben die Elektroniker. Ihr habt ja Recht! Asterix würden sagen: „Die spinnen, die …..

195 High-Pegel entspricht der logischen 1 und muss über 2,5 V sein. Die +3,3 V, die der Raspberry Pi am GPIO liefern kann, reichen also.

196 Low-Pegel entspricht dr logischen 0 und muss under 0,5 V sein. Auch das kann der Raspberry Pi am GPIO.

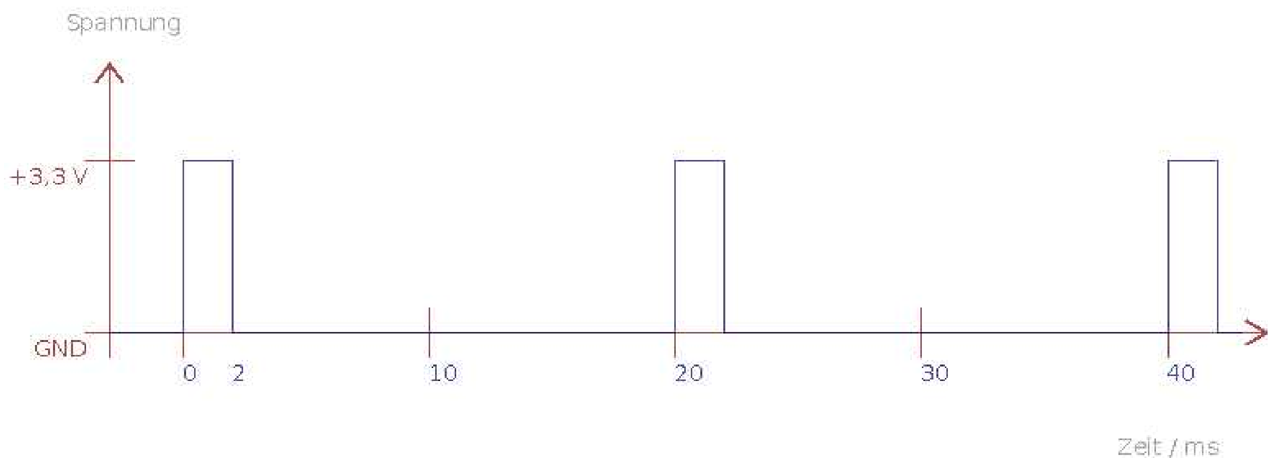


Abbildung 29: Spannungs - Zeit Diagramm für ein Servo PWM-Signal

In Bild 29 habe ich als Beispiel eine Pulsweite von 2 ms eingezeichnet. Das bedeutet also rechter Vollausschlag. Wenn ihr sicher stellen wollt, dass das Servo genug Zeit hat, um in die gewünschte Stellung zu fahren, solltet ihr einige Pulsweiten mit der gewünschten Zeit übermitteln. Falls ihr mehr zu Servos wissen wollt, empfehle ich euch einen Blick auf die Web-Seite: „wiki.rc-network.de/Servo" zu werfen.

10.2 Servos ansteuern mit dem Raspberry Pi

Wir haben gelernt, wie wir LEDs blinken lassen können. Dazu werden in zeitlicher Reihenfolge GPIO-Pins ein- und ausgeschaltet. In dem Zeitdiagramm in Bild sehen wir, dass das Ansteuern von Servos eigentlich genau so geht. Lediglich die Zeiten sind etwas anders. Bei den LEDs war es immer 500 ms an und 500 ms aus. Dazu nutzten wir Mikes c++ Befehl:

```
bcm2835_delay(500);
```

den er uns in seiner Header-Datei „bcm2835.h" zur Verfügung gestellt hatte und übergaben diesem Befehl die Integer Zahl 500, die für 500 ms stand. Ihr erinnert euch. Wir brauchen zum Ansteuern von Servos vier delay-Zeiten: Ungefähr eine Millisekunde für Vollausschlag rechts, ungefähr zwei Millisekunden für Vollausschlag links, etwa 1,5 Millisekunden für die Mittelstellung und 20 Millisekunden für die Periode, mit der die Pulse wiederholt werden sollen. Wir brauchen also delay-Zeiten, die genauer sind als ganzzahlige Vielfache von einer Millisekunde. Wenn wir uns die Header-Datei „bcm2835.h" mit dem Emacs noch mal genauer ansehen, finden[197] wir doch tatsächlich eine Funktion, die uns weiterhilft. Das ist die Funktion:

```
extern void bcm2835_delayMicroseconds (unsigned int micos);
```

Und mehr noch, wir finden auch noch zwei #define Anweisungen an den Compiler, die diese delay-Befehle umbenennen und uns den Bandwurmnamen ersparen. In unserem eigenem c-Programm

197 Wir klicken dazu auf das Lupen-Symbol in der oberen Leiste des Emacs. Das seht schneller, als den ganzen Text zu scrollen.

könne wir also auch ruhig die Befehle:

```
delay(...);
```

und

```
delayMicroseconds(...);
```

verwenden. Aber schauen wir uns die Definition der Funktion Stück für Stück an, um zu verstehen was da abgeht.

Die Definition beginnt mit einem Spezifizierer[198], hier „extern“. Der Emacs hat erkannt, dass es sich dabei um einen Spezifizierer handelt und ihn violett eingefärbt. „extern“ bedeutet, dass die nachfolgende Funktion oder Variable überall verwendet werden kann, also auch außerhalb der Funktion, in der sie vorkommt. Für uns ist das nicht so wichtig, denn wir werden die delay-Funktionen innerhalb der main-Funktion aufrufen.

Es folgt die Angabe des Rückgabe-Typs. Die delay-Funktionen geben aber keinen Wert zurück, sie rechnen ja nichts aus, sie warten nur. Die Bezeichnung eines Typs, den es gar nicht gibt, ist in c/c++ „void“[199].

Innerhalb der Funktionsklammer steht noch „unsigned int“. Dies bedeutet, dass die ganze Zahl, die der Funktion übergeben wird, im Gegensatz zu nur „int“ nicht negativ sein darf. Typenbezeichnungen dürfen bei der Definition von Funktionen in c/c++ Programmen niemals weggelassen werden, da der Compiler sonst nicht weiß, wie viel Speicherplatz er für die Funktion reservieren soll. Ach ja, die Hardware-Nähe von c/c++ lässt wieder grüßen.

Jetzt haben wir Alles, was wir brauchen, um mal ein Servo hin und her zu bewegen und wir legen einen neuen Ordner unter /home/pi/c++Progs an mit dem Namen DrittesProjekt. In diesem Projekt-Ordner erstellen wir einen neuen File mit dem Namen „servo.c“. Am einfachsten machen wir das, indem wir aus dem Zweiten Projektordner den File „dreiLEDsblinken.c“ kopieren und umbenennen. Dann brauchen wir nur die entscheidenden Stellen abzuändern und sparen uns lästige Tipparbeit.

Das Programm ohne die Kommentarzeilen am Anfang seht ihr in Bild 30. Mit den beiden #include-Dateien erweitern wir wieder unsere Möglichkeiten des Sourcecodes. Die delay-Funktionen und die Funktionen für das Ansteuern der GPIO-Pins holen wir uns über die Header-Datei „bcm2835.h“. Die Ausgabefunktion „printf()“ wird durch die Einbindung des Header-Files „stdio.h“ ermöglicht[200]. Den PWM-Eingang am Servo verbinden wir mit Pin 10 vom GPIO-Stecker P1. GND holen wir uns von Pin 6 und die +5 Volt von Pin 2 oder 4.

198 Genauer gesagt ein Typenspezifizierer. Andere Beispiele für Spezifizieren sind zum Beispiel „unsigned“ und „long“.

199 „void“ heißt soviel wie „egal“.

200 Eigentlich brauchen wir keine Ausgabe auf die Konsole, um ein Servo zu bewegen. Es ist aber ganz praktisch zu sehen, ob das Programm überhaupt läuft, falls wir Probleme mit dem richtigen Anschliessen des Servos haben.

emacs23@raspberrypi <@raspberrypi>

File Edit Options Buffers Tools C Help

```
#include <bcm2835.h>
#include <stdio.h>

// Servo an Pin 10
#define PIN RPI_V2_GPIO_P1_10

int main(int argc, char **argv)
{
        // If you call this, it will not actually access the GPIO
        // Use for testing
        // bcm2835_set_debug(1);
    if (!bcm2835_init())
        return 1;

        printf("Hallo Axel\n");

    // Set the pin to be an output
    bcm2835_gpio_fsel(PIN, BCM2835_GPIO_FSEL_OUTP);

    // Beweg das Servo
    int i;
    while (1)
    {
        i=1;
        while (i<100)
        {
                i++;
                bcm2835_gpio_write(PIN, HIGH);
                delayMicroseconds(1800);
                bcm2835_gpio_write(PIN, LOW);
                delay(18);
        }
        i=1;
        while (i<100)
        {
                i++;

                bcm2835_gpio_write(PIN, HIGH);
                delayMicroseconds(1200);
                bcm2835_gpio_write(PIN, LOW);
                delay(18);
        }
    }
    return 0;
}
```

Abbildung 30: Der Sourcecode "servo.c" im Emacs

Unsere Endlosschleife beginnt wieder mit „while(1)“. Davor haben wir eine Zählvariable mit

```
int i;
```

definiert. Als erstes setzen wir die Variable „i“ auf 1. Mit der weiteren while-Schleife fragen wir ab, ob „i“ auch kleiner als 100 ist. Innerhalb dieser while-Schleife zählen wir „i“ mit dem Befehl

```
i++;
```

hoch. Dann setzen wir den Ausgang von Pin 10 für 1800 µs auf +3,3 V und danach für 18 ms auf GND. Das machen die Befehle:

```
bcm2835_gpio_write(PIN, HIGH);
delayMicroseconds(1800);
bcm2835_gpio_write(PIN, LOW);
delay(18);
```

Weil diese while-Scheife etwa 100 mal 20 ms durchlaufen wird, bleibt das Servo etwa 2 Sekunden in der Stellung. Danach folgt eine weitere while-Schleife mit einer Pulsdauer von 1200 µs die auch noch 99-mal durchlaufen wird und das Servo in die andere Richtung dreht.

Dann fängt alles wieder von vorn an und immer so weiter, bis wir Ctrl-C drücken oder den Stecker ziehen[201].

Zu diesem Programm erzeugen wir uns natürlich noch einen geeigneten Makefile und kompilieren das Ganze. Das geht wie beim Blinken der LEDs.

11 Die Ameise lernt laufen

Wir wissen jetzt alles, um unserer Ameise das Laufen bei zu bringen. Ihr könnt also eigentlich schon loslegen und selbst einen Sourcecode schreiben, denn ihr wisst ja jetzt wie das geht. Ich habe das auch getan und euch meinen kompletten Sourcecode im Anhang abgedruckt. Den gibt es aber natürlich auch auf der CD.

Mein Programm arbeitet mit Eingaben, so dass ich auch noch steuern kann, ob die Ameise vorwärts, rückwärts, nach rechts, oder nach links läuft. Hier können wir meinen Sourcecode Stück für Stück meinen Sourcecode einmal durchgehen. Ich habe zunächst einmal einen Kommentarblock geschrieben geschrieben

```
// antV01.c
//
// Program to drive 3 servos using bcm2835 library on a Raspberry Pi
//
// 0        0  0          legs
//  \       |   \
```

201 Den Stecker ziehen wir nur im äußersten Notfall, sonst müssen wir ja wieder booten, und das hält nur auf.

```
//   \      |    \
//   S1-----S2----S3      servos
//     \    |      \
//      \   |       \
//       0  0        0    legs
//
// Servo S1 and S2 are turning right and left
// Servo S3 is turning up and down.
// By this the ant is crowling forward.
//
// After installing bcm2835, you can build this
// with something like:
// gcc -o <filename> -l rt <filename>.c -l bcm2835
// sudo ./<filename>
//
// This software is under GPLv3 Lisence.
// My special thanks to Mike McCauley (mikem@open.com.au)
// Copyright (C) 2013 Axel Huelsmann
```

Dann habe ich die einzubindenden Header-Dateien mit „#include“ angegeben und mir mit „#define“ etwas Tipparbeit erleichtert. Wie ihr seht will ich die drei Servos an Pin 8,10 und 12 anschließen.

```
#include <bcm2835.h>
#include <stdio.h>

// servos are connected to the GPIO P1 connector Pin 08, Pin10 and Pin12.
// GND of the servos are connected to Pin06, +5V supply is connected to Pin02
#define SERVO1 RPI_V2_GPIO_P1_08
#define SERVO2 RPI_V2_GPIO_P1_10
#define SERVO3 RPI_V2_GPIO_P1_12
```

Nachfolgent habe ich mir ein paar Funktionen definiert. Will man in c selbst Funktionen verwenden, muss man die Funktion definieren und kann dann später den Sourcecode dazu schreiben. Das kommt euch erst mal doppelt gemoppelt vor, hat aber den Sinn, das der Compiler sich Speicherplatz reservieren kann.

```
void up(int);
void down(int);
void right(int);
```

```
void left(int);
void turnright(int);
void turnleft(int);
void neutral(int);
```

Jetzt folgt unser Hauptfunktion „main“, die Initialisierung der GPIOs und eine Einzeilige Bedienungsanleitung.

```
int main(int argc, char **argv)
{
  // If you call this, it will not actually access the GPIO
  // Use for testing
  // bcm2835_set_debug(1);
  if (!bcm2835_init())
        return 1;

      printf("g=o b=ack l=eft r=ight\n");
```

Als nächstes Teilen wir der GPIO-Schnittstelle noch mit, dass an PIN 8,10 und 12 Ausgänge haben wollen. Und dann definieren wir noch drei Variablen. Die Variable „speed“ definiert, wie oft unsere PWM-Signale an die Servos wiederholt werden sollen. Das bestimmt die Geschwindigkeit, mit der unsere Ameise krabbelt. Die Variable „i“ nutzen wir wieder für Zähler. Die Variable „key[202]“ nutzen wir zur Eingabe von Buchstaben. Buchstabenvariablen sind vom Typ „char[203]“.

```
    // Set the GPIO pins to be an output
    bcm2835_gpio_fsel(SERVO1, BCM2835_GPIO_FSEL_OUTP);
    bcm2835_gpio_fsel(SERVO2, BCM2835_GPIO_FSEL_OUTP);
    bcm2835_gpio_fsel(SERVO3, BCM2835_GPIO_FSEL_OUTP);

    // start to move
    int speed=50;
    int i=0;
    char key='g';
```

Ab hier startet die Endlosschleife mit „while(1)“. Mit dem Befehl „getchar()“ wartet das Programm so lange, bis eine Taste gedrückt wurde und übergibt die Taste an die Variable „key“. Danach folgt

202 Ich habe den Namen „key“ gewählt, der hier soviel heißt wie „Taste“, weil hier die Variable aus der Tastaturabfrage gespeichert werden soll, mit der ich mein Programm steuern kann.

203 „char“ steht für „character“, auf deutsch „Zeichen“. Variablen vom Typ „char“ benötigen nur ein Byte im Speicher und sind daher die kleinste adressierbare Speichereinheit.

der „switch-Befehl“. Den muss ich euch hier kurz erklären.

Ihr habt bereits die if-Verzweigung kennen gelernt. Zur Erinnerung: Wenn in der runden Klammer hinter dem „if“ etwas wahres stand, dann wurde der Befehl dahinter ausgeführt. Stand da etwas falsches, wurde dieser Befehl ignoriert. Bei der switch-Anweisung wird geschaut, was in der runden Klammer hinter „switch steht und davon wird abhängig gemacht, welche Befehle jetzt ausgeführt werden sollen. Alle Befehle die aufgrund der switch-Anweisung ausgeführt werden sollen, stehen in einem Block, der mir geschweiften Klammern eingerahmt wird. Es ist wieder guter Programmierstil zur besseren Lesbarkeit den Sourcecodes, der Befehle, die zusammen gehören, gleich weit einzurücken.

Die Befehle, die zusammengehören, werden mit der Anweisung „case[204]“ am Anfang und „break;“ am Ende eingerahmt. Falls also der Fall eintritt, das in der Variablen „key“ ein „b“ steht, wird alles zwischen „case 'b':“ und „break“ ausgeführt, sonst werden diese Befehle übergangen. Die switch-Verzweigung wird also noch auf „l“, „r“ und „g“ geprüft. Falls keiner der Buchstaben in der Variablen „key“ steht, wird „default :“ ausgeführt.

```
    while (1)                // this is the main loop
      {
      key = getchar();
      switch(key)
        {
        case 'b':
          left(speed);
          up(speed);
          right(speed);
          down(speed);
          break;
        case 'l':
          left(speed);
          turnright(speed);
          left(speed);
          turnright(speed);
          break;
        case 'r':
          right(speed);
          turnleft(speed);
          right(speed);
          turnleft(speed);
          break;
        case 'g':
          right(speed);
```

204 „case“ heißt „Fall“ und zur im grammatischen Sinne. Die Programmiersprachen-Grammatik wird auch Syntax genannt.

```
          up(speed);
          left(speed);
          down(speed);
          break;
        default:
          neutral(speed);
          break;
        }
      }
    return 0;
}
```

Damit ist mein Hauptprogramm zu ende, was ihr daran erkennt, das die geschlossene geschweifte Klammer der main-Funktion kommt.

Nehmen wir an, in der Variablen „key“ steht 'g' für go[205]. Dann werden nacheinander die Funktionen „right“, „up“, „left“, „down“ ausgeführt. Wenn keine Eingabetaste gedrückt wurde, dann wird durch „default :“ die Funktion „neutral“ ausgeführt.

Würde ich nicht nur einmal die g-Taste drücken habe, sondern „ggg“, so würde die while(1)-Schleife drei mal hintereinander alles was in „case 'g' :“ steht ausführen. Die Ameise würde also drei Schritte nach vorn gehen und dann wieder wegen „default :“ in Neutral-Stellung. Entsprechendes passiert zum Beispiel bei „rrr“ oder „gglggll“. Der Tastaturspeicher merkt sich nämlich die Tasteneingabe und die while(1)-Scheife arbeitet die Befehle an mein Programm nacheinander ab.

Aber schauen wir uns meinen Sourcecode der Funktion „neutral“ mal genauer an. Der Funktion „neutral“ wird die Variable „speed“ übergeben und die Zählervariable „i“ wird auf 0 gesetzt. Dann kommt eine neue while-Schleife, die 50 mal durchlaufen wird[206]. Innerhalb der while(i<50)-Schleife wird die Variable „i“ mit dem Befehl „i++;“ bei jedem Durchlauf um eins erhöht. Das vordere und hintere Servo unserer Ameise sind an SERVO1 und SERVO2 angschlossen. Da die Funktion „delayMicroseconds() 1370 µs wartet, gehen die Servos in Mittelstellung.

Ja Moment, war nicht die Mittelstellung der Servos 1500 µs? Ihr habt ja vollkommen Recht. Ich war etwas nachlässig beim beim Montieren meiner Beinchen auf den Servos. Im Nachhinein habe ich etwas mit den Mikrosekunden gespielt, bis es gestimmt hat. Ihr werdet wahrscheinlich ähnliches machen müssen. Bei mir war die Neutralstellung bei beiden Servos gleich, sonst müsste ich das in diesem Sourcecode auch noch berücksichtigen.

Der Funktion „neutral“ ist es übrigens egal was Servo 3 mit den mittleren Beinchen macht. Servo 3 bekommt kein PWM-Signal und ich nutze einfach Isaac Newtons[207] Schwerkraft, um alle 6 Beinchen auf dem Boden zu halten.

---

205 „go“ für gehe!

206 Denn „speed“ war ja 50 und wenn „i“ auch 50 wird, wird die Aussage (i<50) falsch und die Schleife bricht ab.

207 Isaac war genial. Mit der von ihm aufgestellten Gravitationstheorie von 1686 konnte erstmals die Schwerkraft richtig erklärt werden. Die dazu notwendige Mathematik erfand er gleich mit, behauptete er. Wir vermuten heute allerdings, das er von Königin Charlottes Freund, Gottfried Wilhelm Leibniz (nein, nicht der mit dem Keks) abgeschrieben hat.

```
void neutral(int speed)
{
  int i=0;
  while (i<speed)
  {
    i++;
    bcm2835_gpio_write(SERVO1, HIGH);
    bcm2835_gpio_write(SERVO2, HIGH);
    delayMicroseconds(1370);
    bcm2835_gpio_write(SERVO1, LOW);
    bcm2835_gpio_write(SERVO2, LOW);
    delay(15);
  }
}
```

Die Funktion „right“ ist nach dem selben Muster aufgebaut. Wie ihr seht, werden jetzt auch alle drei Servos angesteuern. Servo 1 und 2 drehen nach rechts, während Servo 3 runter geht. Auch hier habe ich die Mikrosekunden von Servo 3 nachjustiert.

```
void right(int speed)
{
  int i=0;
  while (i < speed)
    {
      i++;
      bcm2835_gpio_write(SERVO1, HIGH);
      delayMicroseconds(1000);
      bcm2835_gpio_write(SERVO1, LOW);
      bcm2835_gpio_write(SERVO2, HIGH);
      delayMicroseconds(1000);
      bcm2835_gpio_write(SERVO2, LOW);
      bcm2835_gpio_write(SERVO3, HIGH);
      delayMicroseconds(1250);
      bcm2835_gpio_write(SERVO3, LOW);
      delay(15);
    }
}
```

Die Funktione „up“, „left“ und „down“ laufen nach dem gleichen Muster ab. Ich muss dazu nicht mehr viel erklären, oder?

```
void up(int speed)
{
  int i=0;
  while (i < speed)
    {
      i++;
      bcm2835_gpio_write(SERVO1, HIGH);
      delayMicroseconds(1000);
      bcm2835_gpio_write(SERVO1, LOW);
      bcm2835_gpio_write(SERVO2, HIGH);
      delayMicroseconds(1000);
      bcm2835_gpio_write(SERVO2, LOW);
      bcm2835_gpio_write(SERVO3, HIGH);
      delayMicroseconds(1600);
      bcm2835_gpio_write(SERVO3, LOW);
      delay(15);
    }
}

void left(int speed)
{
  int i=0;
  while (i < speed)
    {
      i++;
      bcm2835_gpio_write(SERVO1, HIGH);
      delayMicroseconds(1700);
      bcm2835_gpio_write(SERVO1, LOW);
      bcm2835_gpio_write(SERVO2, HIGH);
      delayMicroseconds(1750);
      bcm2835_gpio_write(SERVO2, LOW);
      bcm2835_gpio_write(SERVO3, HIGH);
      delayMicroseconds(1600);
      bcm2835_gpio_write(SERVO3, LOW);
      delay(15);
```

```
        }
    }

    void down(int speed)
    {
      int i=0;
      while (i < speed)
        {
          i++;
          bcm2835_gpio_write(SERVO1, HIGH);
          delayMicroseconds(1700);
          bcm2835_gpio_write(SERVO1, LOW);
          bcm2835_gpio_write(SERVO2, HIGH);
          delayMicroseconds(1750);
          bcm2835_gpio_write(SERVO2, LOW);
          bcm2835_gpio_write(SERVO3, HIGH);
          delayMicroseconds(1250);
          bcm2835_gpio_write(SERVO3, LOW);
          delay(15);
        }
      }
```

Ich hatte anfangs Schwierigkeiten eine Drehbewegung hin zu bekommen, bis ich die Funktionen „turnright“ und „turnleft“ eingeführt habe. Dann habe ich das mittlere Servo 3 einfach in der Ansteuerung weg gelassen und dann ging's.

```
    void turnright(int speed)
    {
      int i=0;
      while (i < speed)
        {
          i++;
          bcm2835_gpio_write(SERVO1, HIGH);
          delayMicroseconds(1000);
          bcm2835_gpio_write(SERVO1, LOW);
          bcm2835_gpio_write(SERVO2, HIGH);
          delayMicroseconds(1000);
          bcm2835_gpio_write(SERVO2, LOW);
          delay(15);
```

```
    }
  }



  void turnleft(int speed)
  {
    int i=0;
    while (i < speed)
      {
        i++;
        bcm2835_gpio_write(SERVO1, HIGH);
        delayMicroseconds(1700);
        bcm2835_gpio_write(SERVO1, LOW);
        bcm2835_gpio_write(SERVO2, HIGH);
        delayMicroseconds(1750);
        bcm2835_gpio_write(SERVO2, LOW);
        delay(15);
      }
    }
```

Mein Programm ist für euch natürlich nur ein Beispiel, um erst mal anzufangen. Ihr müsst zum Compilieren wieder einen Makefile schreiben, aber ihr wisst ja jetzt wie das geht.

Um die richtigen Mikrosekunden zu finden, habe ich meine Ameise aufgebockt, sonst reißt mir das Mistvieh immer die Leitungen raus. Frei laufen kann die Ameise so noch nicht, denn ich habe immer noch Tastatur, Maus und Monitor an den Raspberry Pi angeschlossen. Aber das führt uns zum nächsten Kapitel.

12 Die Ameise fernsteuern mit ssh

Eine super praktische Sache, um von einem Rechner einen anderen fernzusteuern, ist SSH[208]. SSH ermöglicht eine verschlüsselte Verbindung zwischen zwei Rechnern über ein unsicheres Netzwerk. Der eine Rechner ist der Raspberry Pi, der andere Rechner ist ein PC oder Laptop. Wie wir den Raspberry Pi über den WLAN-Haus-Router ans Internet bringen haben wir schon besprochen. Der PC sollte auch eine Verbindung zu dem Haus-Router haben. Dabei ist es egal, ob über WLAN oder über Ethernet-Kabel. Schön wäre es, wenn der PC oder Laptop ein LINUX-Betriebssystem hat, denn damit kennen wir uns ja jetzt schon ein bisschen aus. Apple ist wie LINUX ein Unix-artiges Betriebssystem und geht wahrscheinlich auch. Ich hab es aber nicht ausprobiert. Falls wir keinen LINUX-PC haben, gibt es zwei Möglichkeiten.

1) Wir installieren LINUX als zweites Betriebssystem[209] .

---

208 SSH steht für secure shell, sichere „shell“. Was eine „shell“ ist wisst ihr noch von der „bash“.
209 Ich bevorzuge OpenSuse mit KDE. OpenSuse kann man sich im Zeitschriftenhandel als Beilage-CD besorgen, dann hat man auch gleich eine Installationsanleitung, oder man lädt es bei www.opensuse.org direkt herunter und brennt

2) Wir verwenden das Programm PuTTY unter Windows.

Auf dem Raspberry Pi haben wir ja schon dafür gesorgt, das ssh läuft. Falls nicht, schauen wir uns das noch mal in Kapitel 4.10 auf Seite21 an.

12.1 PuTTY unter Windows

Diese Kapitel kann überschlagen werden, wenn wir einen LINUX-PC zur Verfügung haben. Bei einem Windows-PC installieren wir zunächst das Programm PuTTY. Um sicher zu stellen, dass wir nicht einem Freeware-Piraten[210] auf dem Leim gehen, gehen wir folgendermaßen vor:

1) Wir gehen mit dem InternetExplorer oder dem Firefox auf die Webseite de.wikipedia.org.
2) Dort geben wir im Suchfeld PuTTY ein. Wikipedia gibt häufig einen Web-Link zu den Originalseiten im Internet an. Das ist sicherer als über Google zu gehen,da es nur eine Möglichkeit gibt. Diese kann zwar auch gefälscht sein, doch das kommt sehr selten vor, und wird außerdem von der Comunity schnell gemerkt und korrigiert.
3) Von dort laden wir uns das Installationsprogramm „putty-0.62-installer.exe“ herunter. Es könnte inzwischen sein, das es inzwischen eine neuere Version als 0.62 gibt, dann laden wir natürlich die.
4) Wir starten das Installationsprogramm und halten uns an die Anweisungen des Installationsprogramms. Am Besten beantwortet man die Fragen immer mit dem Vorschlag, der gemacht wird.
5) Zum Schluss werden wir noch gefragt, ob wir ein Icon auf dem Desktop haben wollen. Wir setzen das entsprechende Häkchen, denn das ist ja ganz praktisch.
6) Wir löschen das Installationsprogramm „putty-0.62-installer[211]“, denn wir wollen uns ja nicht zumüllen.
7) Nun starten wir das Programm PuTTY.

---

es sich selbst auf CD. Das dauert aber ein bisschen. Langfristig kommt ihr als Roboter-Programmierer sowieso nicht an LINUX vorbei, da ihr die Hardware eines Rechners dann leichter voll ausnützen könnt. Daher könnt ihr auch jetzt damit anfangen.

210 Unter Freeware-Piraten verstehe ich Leute, die eigentlich sonst kostenlose Software auf ihrem Server verteilen. Liest man sich nicht alles richtig durch, was auf der Seite steht, und klickt dann unbedarft auf „download“, hat man plötzlich einen Vertrag abgeschlossen und ein Anwalt bedroht einen mit Abmahnung und erheblichen Kosten.

211 Meistens hat sich ein Icon des Installationsprogramms auf dem Desktop eingenistet. Das brauche wir nicht mehr.

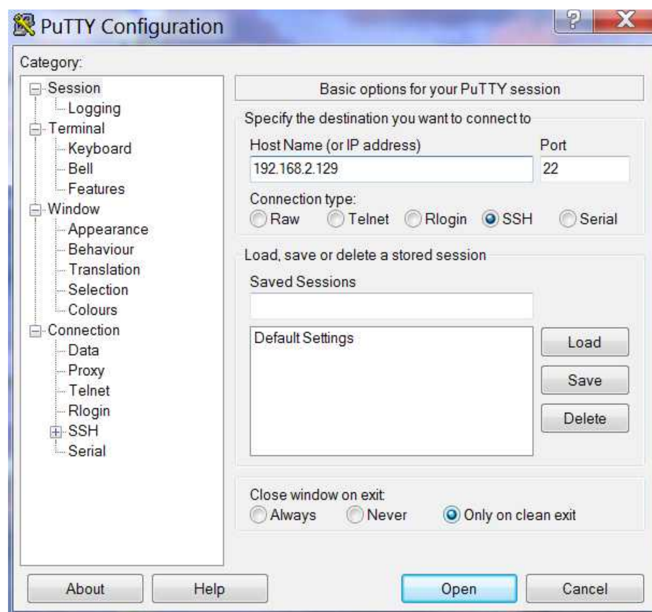


Abbildung 31: Das Konfigurationsfenster von PuTTY

Wenn wir PuTTY starten, meldet sich zuerst der „PuTTY Configurator“ wie in Bild 31 zu sehen ist. Das ist ein Fenster in das wir die IP Adresse unseres Raspberry Pi in das Feld mit der Überschrift “Host Name (or IP address)“ eingeben müssen[212]. Lokale Internetadressen fangen meistens mit 192.168 ….. an. Wir klicken nun unten auf den Button „open“.

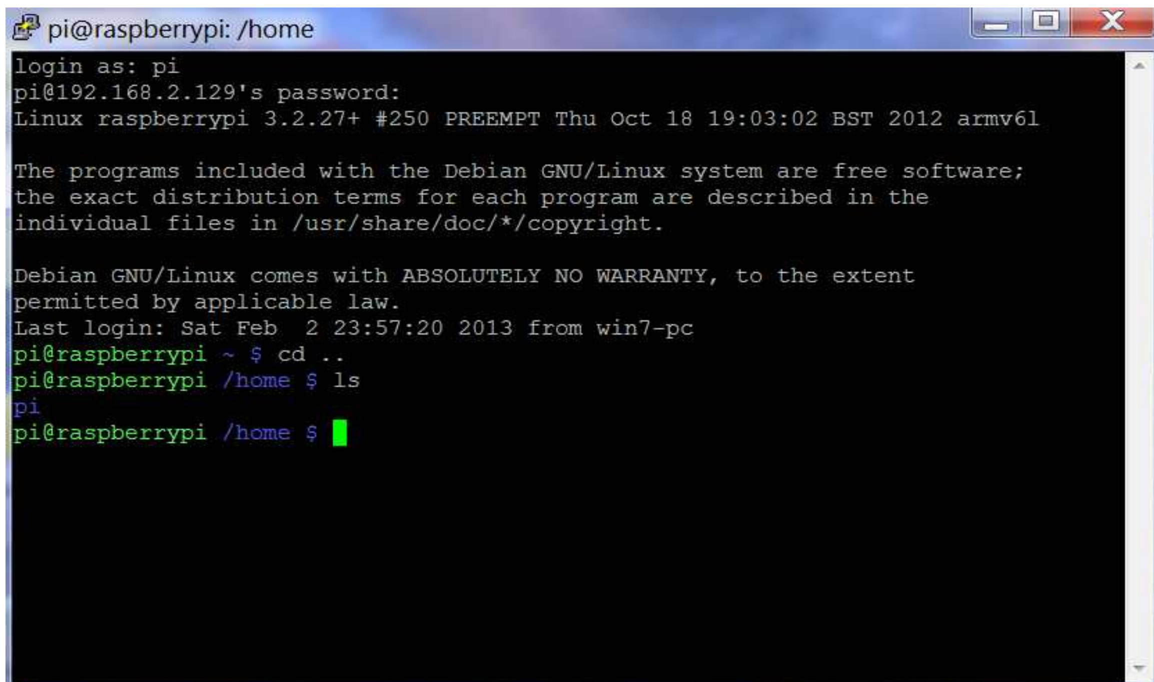


Abbildung 32: Das Terminal Fenster von PuTTY

Der „PuTTY Configurator“ schließt sich nun und wir sehen ein Terminal Fenster 32, in dem wir

212 Wenn wir die IP-Adresse des Raspberry Pi nicht mehr kennen, geben wir im LXTerminl den Befehl „iwconfig eine. Unter wlan0 finden wir dann in der zeiten Zeile: „inet addr: 192.168.2.xxx“.

aufgefordert werden, unseren Benutzernamen auf dem Raspberry Pi einzugeben mit den Worten: „login as:“. Hier geben wir natürlich „pi“ ein. Jetzt werden wir nach dem Passwort auf unserem Raspberry Pi gefragt mit den Worten: „pi@192.168.x.yyy's password:“ Hier geben wir natürlich „raspberry[213]“ ein.

Und siehe da, nach mehreren Zeilen Text, den ihr ja mal übersetzen könnt, taucht der Prompt des Raspberry Pi auf. Wir gehen aus purer Freude mal den Befehl „ls -l“ ein, und die jetzige Meldung sollte uns irgendwie bekannt vorkommen.

12.2 SSH unter LINUX

12.3 Die Ameise zum Leben erwecken

Nachdem wir mit ssh oder PuTTY ein Verbindung zum Raspberry Pi aufgenommen haben, geben wir den Befehl „sudo shutdown now“ ein, um den Raspberry Pi herunter zu fahren und aus zu schalten. Das dauert ein Weilchen. Ist alles aus, der Monitor vom Raspberry Pi dunkel und die Verbindung zu ihm erloschen, können wir jetzt unsere Roboter-Ameise von den Kabeln zu Maus, Monitor und Tastatur befreien. Nur den WLAN-USB-Adapter sollten wir noch eingesteckt lassen. Wir versorgen jetzt den Raspberry Pi über die Batterie der Roboter-Ameise.

Sobald der Raspberry Pi an die Batterie angeschlossen ist und wieder Strom hat, sollte er wieder booten. Das können wir jetzt leider nur noch an der blinkenden grünen LED mit der Bezeichnung „ACT“ erkennen. Nach einer Weile sollte auch der WLAN-USB-Adapter seine Arbeit aufnehmen und uns das mit blinken mitteilen. So ein, zwei Minuten kann es durchaus dauern, bis der Raspberry Pi wieder da ist und wir zu ihm Verbindung aufnehmen können. Jetzt natürlich über den Windows-PC mit PuTTY oder über den LINUX-PC mit ssh.

Haben wir wieder Verbindung, können wir jetzt mit „cd“ in unser Verzeichnis mit dem Programm „antv01“ wechseln und dieses wieder mit „sudo ./antV01“ starten. Mit der Tastatur des PC können wir jetzt die Roboter-Ameise zum Beispiel mit „ggglllllggg“ über WLAN fern steuern. Viel Spass dabei.

13 Eine Kamera anschließen

14 Ein LINUX-Labtop als Fernsteuerung

- hostapd auf dem Labtop installieren. Macht den Labtop zum Access Point.
- anlegen der Datei /etc/hostapd_minimal.conf mit Emacs

```
# test hostapd
inferface=wlan0
driver=nl80211
ssid=axel2rpi
channel=6
```

213 Während wir das Passwort eingeben, sehen wir nicht was wir geschrieben haben. Das ist normal, denn andere (die uns über de Schulter gucken) sollen unser Passwort nicht unbedingt mitlesen können.

– Befehl eingeben: hostapd -dd hostapd-minimal.conf

15 Den Staub unterm Bett suchen

Bla bal

16 Anhang

16.1 Liste der Komponeten

Raspberry Pi

SDHC-Speicherkarte

USB-Tastatur

USB-Maus

USB-Hub mit Netzteil

HDMI-Kabel zum Monitor

USB-WLAN Adapter

Grundplatte

Servo

Federstahl-Draht 2mmfür die Beine

Holzperlen für die Füße

Flachbandkabel 26-polig

Stiftleiste 2x13, 2,53 mm Raster

26-polige Buchse für Flachbandkabel

Batteriekasten

NiMH-Akkus AA

Batterie Klip

Schalter

Distanzstücke

Schrauben

Bindedraht

USB-Kamera

16.2 LINUX-Befehle

info, help, man

ls; ls -l

cd; cd ..

mv

rm

top

mkdir

rmdir

sudo

gcc

w

i Linus Torvalds, David Diamond, Just for Fun: Wie ein Freak die Computerwelt revolutionierte. Die Biographie des LINUX-Erfinders. Deutscher Taschenbuch Verlag, 2002, **ISBN-10:** 3423362995

ii Michael Kofler, LINUX – Studentenausgabe, Addison-Wesley Verlag, 2009